Reliable Data Delivery in Large-Scale Low-Power Sensor Networks

DANIELE PUCCINELLI
University of Applied Sciences of Southern Switzerland
and
MARTIN HAENGGI
University of Notre Dame

In data collection applications of low-end sensor networks, a major challenge is ensuring reliability without a significant goodput degradation. Short hops over high-quality links minimize per-hop transmissions, but long routes may cause congestion and load imbalance. Longer links can be exploited to build shorter routes, but poor links may have a high energy cost. There exists a complex interplay among routing performance (reliability, goodput, energy efficiency), link estimation, congestion control, and load balancing; we design a routing architecture, Arbutus, that exploits this interplay, and perform an extensive experimental evaluation on testbeds of 100-150 Berkeley motes.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms: Algorithms, Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Wireless sensor networks, routing, load balancing, congestion control

ACM Reference Format:

Puccinelli, D. and Haenggi, M. 2010. Reliable data delivery in large-scale low-power sensor networks. ACM Trans. Sensor Netw. 6, 4, Article 28 (July 2010), 41 pages.
DOI = 10.1145/1777406.1777407 http://doi.acm.org/10.1145/1777406.1777407

This article builds on previously published results: Arbutus: Network-Layer Load Balancing for Wireless Sensor Networks, published at the IEEE Wireless Communications and Networking Conference (WCNC'08), and DUCHY: Double Cost Field Hybrid Link Estimation for Low-Power Wireless Sensor Networks, published at the 5th Workshop on Embedded Networked Sensors (HotEmNets'08).
This research was funded by NSF (CNS 04-47869), DTRA (N00164-07-8510), and the EU-NEST project MEMORY (FP6 EU-NEST 43236).
Authors' addresses: D. Puccinelli, University of Applied Sciences of Southern Switzerland, Galleria 2, Via Cantonale, CH-6928 Manno, Switzerland; email: daniele.puccinelli@supsi.ch; M. Haenggi, Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556.

DOI 10.1145/1777406.1777407 http://doi.acm.org/10.1145/1777406.1777407

1. INTRODUCTION

Due to their resource limitations, low-power Wireless Sensor Networks (WSNs) pose considerable communication challenges. One of the most significant is preventing packet loss while maintaining an acceptable goodput. Aside from catastrophic problems such as hardware or software failure, packet loss may occur due to channel errors, congestion-induced buffer overflow, and protocol-level inefficiencies.

Wireless propagation effects such as large-scale path loss, shadowing, and multipath fading contribute to the attenuation of the signal power. In Zhao and Govindan [2003], Woo et al. [2003], and Zuniga and Krishnamachari [2007], it is shown that a *transitional reception region* separates a region with high connectivity from a disconnected region, and asymmetric links are shown to be common in the transitional region: wireless connectivity is neither Boolean nor bidirectional. Differently from high-end wireless networks such as WLANs or mobile ad hoc networks (MANETs), low-power WSNs typically employ low-end transceivers with a low maximum transmit power (typically 0dBm), and are therefore completely exposed to the vagaries of RF propagation. Link estimation is instrumental in limiting channel-related packet loss and minimizing the number of retransmissions. Channel-related losses are also caused by interference: for instance, CSMA-based MAC layers, common in WSNs, are exposed to hidden node effects. Congestion is particularly severe in WSNs due to their typical many-to-one traffic pattern, which may lead to buffer overflow depending on the network topology. Network protocols may also be responsible for additional losses, for instance, due to routing loops and egress drops (the elimination of packets that are erroneously believed to be flawed, such as false duplicates). Incompatibility between protocols pertaining to different layers may also be conducive to a significant performance degradation. An example is the use of a network protocol that requires promiscuous mode operation for link estimation along with a MAC protocol that avoids snooping to save energy [Langendoen et al. 2006].

Many routing solutions have been proposed in the literature, but only a handful of them have been implemented and tested on low-end nodes. The most strenuously tested and heavily used solutions are distributed tree-based schemes that target homogeneous networks. In particular, the MintRoute family [Woo et al. 2003] has formed the core of the TinyOS [Hill et al. 2000] network layer over the past years and has recently led to the Collection Tree Protocol (CTP) [Gnawali et al. 2009].

In this article, we focus on the same corner of the routing design space as the MintRoute family, and we propose Arbutus[1], a routing architecture for data collection applications of low-power WSNs that seeks to achieve high reliability as well as to maximize the goodput given the reliability constraint. The main principle behind the Arbutus architecture is that routing over a few long hops can be much more efficient than routing over many short hops [Haenggi and Puccinelli 2005; Wang et al. 2006]. By *long hops*, we do not

[1]The name *Arbutus* whimsically refers to an evergreen tree, suggesting that Arbutus builds a routing tree that does not lose its leaves.

mean *higher transmit power*: the transmit power is assumed to be the same independently of whether a hop is long or short. The hop length is completely determined by the physics of wireless propagation and is qualitatively said to be short or long compared to the hop length expected on the basis of the large-scale path loss. Due to a particularly favorable (or unfavorable) fading state, a hop may be significantly longer (or shorter) than expected from the large-scale path loss [Puccinelli and Haenggi 2006a]. Routing over many short hops means always minimizing the large-scale path loss, while routing over fewer long hops means leveraging on positive fading states.

The Arbutus architecture employs existing tools and recent results along with two main elements of novelty: a tree construction scheme built into the link estimation level that represents the centerpiece of the architecture and provides a practical way to enforce long-hop routing, and the treatment of congestion control as a first-order problem. Other key contributions include extensive experimental evidence about the advantages of long-hop routing (that complements our work in Haenggi and Puccinelli [2005]), and the investigation of the impact of load balancing on the routing performance (that complements our previous work on the lifetime benefits of load balancing [Puccinelli and Haenggi 2008a, 2009]). We also make a key contribution to the methodology of WSN routing research: while most studies treat the network topology as given and fixed, we treat it as a parameter, and provide ample experimental evidence about the importance of this choice. We also begin the study of the impact of the network topology by providing quantitative guidelines for its experimental analysis.

Arbutus has been implemented in TinyOS 2.x on top of the standard CSMA-based MAC layer. We adopt an experimental approach and evaluate Arbutus on large-scale networks of 100–150 nodes using remote-access testbeds at Harvard University (MoteLab [Werner-Allen et al. 2005]), the Technische Universitaet Berlin (Twist [Handziski et al. 2006]), and the University of Southern California (Tutornet[2]). We benchmark Arbutus against CTP [Gnawali et al. 2009], the strenuously tested reference routing protocol for TinyOS 2.x. CTP has been shown to work well in mote networks and has already been used in several studies [Choi et al. 2007a, 2007b; Wachs et al. 2007; Fonseca et al. 2007; Hauer et al. 2008; Kothari et al. 2008; Filipponi et al. 2008].

2. BACKGROUND AND RELATED WORK

2.1 Generalities

For ease of reference, Table I provides a list of the symbols and variables used in this article (each symbol is also described in the text).

Nodes and sets thereof. We denote the set of all nodes in a given network by $\mathcal{N} \subset \mathbb{N}$; for simplicity, we will refer to $\mathcal{N}$ as the network, but we intend it to only denote the set of nodes (and not the set of links). We use the symbol z for an invalid node. We assume the presence of only one sink in the network, denoted

[2]http://enl.usc.edu/projects/tutornet

Table I. List of the Main Symbols Used in the Article

Nodes and sets thereof	
$\mathcal{N}$	set of nodes in the network
$\mathcal{N}_1$	sink neighborhood
s	sink
z	invalid node
$p(i)$	parent of node i
$\tilde{p}(i)$	former parent of node i
$p_a(i)$	advertised parent of node i
Load	
f_{gen}	target data packet generation rate (target offered load)
β_i	relayed load at node i (in pkts/sec)
Links	
(i, j)	physical link between nodes i and j
$[i, j]$	generalized link between nodes i and j
$[i, s]_k$	generalized link between nodes i and the sink with a first hop over (i, k)
$\pi_{i,j}$	Packet Delivery Rate (PDR) over (i, j)
M_i	Required Number of Packets (RNP) with no regard to the identity of $p(i)$
$R_{i,k}$	Total Required Number of Packets (RNP) over (i, k), with $p(i) = k$
$E_{i,j}$	Expected Number of Transmissions (ETX) over (i, j)
e_i	delivery ratio of node i measured at the sink
Data plane: Retransmissions	
$N_{\max}$	maximum number of retransmissions at fixed intervals
T_r	fixed inter-transmission interval
T_v	variable inter-transmission interval
Data plane: Congestion control	
q	fraction of occupancy of the data FIFO buffer
W_{c}	congestion threshold
Control plane: Outer cost field	
H_i	hop count between node i and the sink
A_i	outer cost at i
c	binary depth correction
Control plane: Inner cost field	
$\text{nCSI}_{i,j}$	normalized CSI over (i, j)
$\text{nRSS}_{i,j}$	normalized RSS over (i, j)
$\text{nLQI}_{i,j}$	normalized LQI over (i, j)
Λ_i^{nCSI}	Normalized CSI bottleneck function of node i
Λ_i^{nRSS}	Normalized RSS bottleneck function of node i
Λ_i^{nLQI}	Normalized LQI bottleneck function of node i
w_{CSI}	Binary selection variable for the CSI terms in the inner cost metric
w_{RSS}	Binary selection variable for the RSS terms in the inner cost metric
w_{LQI}	Binary selection variable for the LQI terms in the inner cost metric
w_{B}	Binary selection variable for the load term in the inner cost metric
B_i	load bottleneck function of node i
C_i^b	inner cost at i if $p(i) = b$
C_i	inner cost at i
Other performance indicators	
ρ	coverage target value
ϕ	fairness

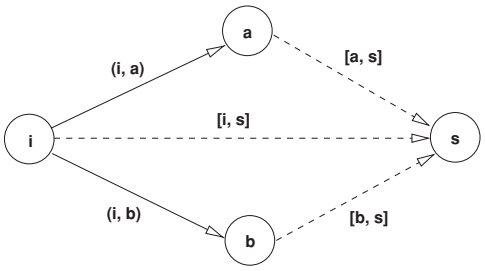


Fig. 1. A distributed routing example viewed from the perspective of a given node i. The arrows show the direction of unicast data traffic. Solid arrows represent physical links, while dashed ones represent generalized links.

as $s \in \mathcal{N}$ (this is not a limitation of Arbutus, which can handle multiple sinks). The application model employed in the experiments in this article is continuous many-to-one data collection: every node generates data packets with target rate f_{gen} and routes them to s. The parent of a node i, denoted as $p(i)$, is the destination of the data packets unicast by i.

Load. We define the *target offered load* f_{gen} as the number of packets that each node attempts to inject into the network per time unit, barring impediments due to congestion control schemes; f_{gen} is assumed to be the same for all nodes. We define the *relayed load* β_i of node i as the number of relayed packets per time unit. Both the target offered load and the relayed load are measured in pkts/sec.

Links. We denote a directed wireless link from node i to node j as (i, j). This notation indicates that the link is *physical*: if i transmits packets using a given physical layer and a set transmit power, j receives at least one of the packets over a given time window T (or else, (i, j) is said not to exist within T). Node j is said to be i's *neighbor* if (i, j) and (j, i) exist.

Generalized links. We define the concept of *generalized link*, which we indicate as $[i, j]$, to represent a route between i and j whose intermediate relays may be unknown. We denote as $[i, s]_k$ the sequence of all the relays used by the routing protocol (at a given time) to deliver i's packets to s using k as the first relay. Figure 1 clarifies our notation using a distributed routing example viewed from the perspective of a given node i. Node i wants to send a packet to the sink s. A distributed routing protocol runs at every node, allowing each node to know the address of the next hop toward the sink. Let us visualize the network from i's point of view: i has physical links to its neighbors a and b (they can all hear each other directly). Node i, however, does not know what lies beyond its neighbors, other than the fact that the sink s is somewhere downstream from them. Therefore, we say that i and s are connected by a generalized link, $[i, s]$. If $p(i) = a$, then $[i, s]$ corresponds to $[i, s]_a$. Likewise, if $p(i) = b$, then $[i, s]$ corresponds to $[i, s]_b$. It is up to the distributed routing protocol to determine whether a or b should be $p(i)$. A node is said to have a valid route to the sink if all nodes along the route have a valid parent, that is, if $p(k) \neq z$ for all $k \in [i, s]$.

From a node's point of view, a *route breakage* is equivalent to the lack of a valid parent ($p(i) = z$).

Link performance measures. We define the Packet Delivery Rate (PDR) over (i, j), $\pi_{i,j}$, as the ratio between the number of packets received by j over the number of packets sent by i to j over a given time window. The PDR is therefore an empirical probability of reception over (i, j). An *asymmetric link* is defined in this article as a link (i, j) with $|\pi_{i,j} - \pi_{j,i}| > 0.5$. The delivery ratio at the sink for node i, defined as the number of packets from node i that are delivered to the sink (either directly or through multiple hops) over the total number of packets generated by node i, is denoted as e_i.

Another link performance measure is the Required Number of Packets (RNP) [Cerpa et al. 2005], which indicates the total number of transmissions that are needed by node i to get a packet across $(i, p(i))$ (averaged over a given time window). Differently from the PDR, the RNP is not necessarily tied to a specific link, as $p(i)$ may change while the same packet is being retransmitted. Indeed, if a given parent requires too many retransmissions, parent rotation may be encouraged by the network layer (parent changes dictated by a routing protocol are whimsically known as *churn*). We employ the notation M_i to indicate the RNP over $(i, p(i))$, deliberately omitting $p(i)$ in the notation because it may change over time, and we denote the RNP for a specific parent $p(i) = k$ as $R_{i,k}$. For a given packet, $M_i = R_{i,k}$ if $p(i) = k$ until the packet's delivery to k, or else $M_i \geq R_{i,k}$.

Link performance estimates. Link performance estimates rely on Channel State Information (CSI). Common forms of CSI include PDR estimates based on control beacon sequence numbers [Woo et al. 2003], the Received Signal Strength (RSS), and the Link Quality Indicator (LQI). Beacon-based PDR estimates are completely platform independent, RSS is made available by most radios, and LQI is only available with 802.15.4-compliant devices. The RSS used to be considered a poor predictor of link quality, mostly because of the limitations of early platforms [Zhao and Govindan 2003]. The community has recently focused on the 802.15.4 stack, and, in particular, on motes built around the CC2420 transceiver, which provides a much more reliable RSS indication. The RSS is a good predictor of link quality; specifically, it has been shown that the RSS, if higher than about −87dBm, correlates very well with the PDR [Srinivasan and Levis 2006]. While most radios provide a RSS indicator, the LQI is specific to 802.15.4, and its implementation is left to the radio chip designer. In the CC2420, the LQI is implemented as the sum of the first 8 correlation values after the start of frame delimiter represented as a 7-bit unsigned integer value.

Critical set. The neighborhood of the sink, $\mathcal{N}_1 = \{i : \pi_{i,s}\pi_{s,i} > 0\}$, will be referred to as the *critical set*, as it contains the *critical nodes* that must relay all upstream traffic to the sink. It is the extra workload that makes these nodes critical: their batteries get drained at a faster rate than their upstream peers, and their energy depletion disconnects the sink from the rest of the network.

Node degree. It is normally defined in the presence of Boolean connectivity as the number of links that a node has to other nodes. In our case, we must take both soft connectivity and link directionality into account, so we define the incoming soft degree of node i as $\sum_{k\in\mathcal{N}} \pi_{k,i}$ and the outgoing soft degree as $\sum_{k\in\mathcal{N}} \pi_{i,k}$. We define the average node degree as the average of the incoming and the outgoing soft degree. The soft degree is therefore an empirical estimate of the expected node degree in the connectivity graph.

2.2 A Network Layer for Sensor Networks

A taxonomy of routing protocols for sensor networks. Various classifications of routing protocols are possible. Depending on whether geographic information is employed or not, we speak of *geographic routing* or *cost-based routing* [Poor 2000]. In the latter, the central idea is the generation of a cost field rooted at the sink. The cost field is set up as nodes estimate the cost of reaching the destination according to a given metric, for example, the distance in number of hops, and packets are routed to the destination through reverse path routing (data packets descend the cost field from the sensing area to the destination). Arbutus does not employ geographic information and is therefore cost-based.

If routing decisions are taken locally at each node, the protocol is *distributed*, whereas it is said to be *centralized* if all decisions are taken by one special node, typically a high-end node. Arbutus is distributed, because it targets homogeneous low-end networks where no single node has sufficient computing resources and energy to support a centralized approach.

Routing schemes are said to be *point-to-point* if they allow any node to route data to any other node. This *any-to-any* routing paradigm, however, is not needed in the largest class of sensor network applications, data collection, where a collection tree (*many-to-one* or *many-to-few*) is normally employed; this is also the case with Arbutus.

Protocols are said to be *sender-based* if nodes have specific parents to which they unicast packets; this is the case with Arbutus. In *receiver-based* routing, a node estimates the cost of reaching the intended destination and includes it in the outgoing packet before broadcasting it. Only the neighbors that estimate a lower cost to the destination rebroadcast the packet, allowing it to descend a loop-free gradient towards the destination. The main problem with the receiver-based approach is the large overhead due to redundant forwarding [Woo 2004].

Multipath routing schemes [Ganesan et al. 2002] distribute traffic over different paths (either alternating the use of different paths at different times, or using multiple paths at the same time). Given the energy cost of redundant transmissions, Arbutus does not follow a multipath approach.

Routing schemes can be classified based on when route selection is performed. In *proactive* protocols, routes are discovered before they are needed, whereas in *reactive* protocols routes are discovered on demand. In the case of cost-based data collection, one could argue that routing is reactive, as the sink initiates it by starting the setup of a cost field. One could also argue, however,

that routing is proactive, because the nodes always know their next-hop neighbor thanks to periodic beaconing.

Most WSN routing protocols are based on collection trees. MintRoute, MultiHopLQI[3], and the Collection Tree Protocol (CTP) [Gnawali et al. 2009] represent successive evolutions of a common cost-based paradigm defined in Woo et al. [2003], which recognizes that the volatility of the wireless channel makes Boolean connectivity models unsuitable to low-end sensor networks with low-power radios and limited resources. In the MintRoute family, a link estimator assesses link quality to help a routing engine choose the neighbor that offers the best progress toward the sink (according to a given metric), and a forwarding engine unicasts local and upstream traffic to the neighbor chosen by the routing engine.

Long-hop and short-hop routing. For multihop wireless networks, a fundamental question is whether it is better to route over many short hops (short-hop routing) or over a smaller number of longer hops (long-hop routing). In Haenggi and Puccinelli [2005], we have shown 18 reasons why long-hop routing is, in most cases, a very competitive strategy: less energy consumption, better load balancing, a more aggressive exploitation of radio sleep modes, and a reduced route maintenance overhead are the most relevant for low-end nodes. Routing over a few long hops is efficient if lossy links can be avoided through robust link estimation. In Couto et al. [2003], the Expected Number of Transmissions (ETX) metric is proposed; the idea is to estimate the total number of transmissions needed to get a packet across a link, and use the route with the minimum ETX. For (i, j), the ETX is estimated as $E_{i,j} \approx 1/(\pi_{i,j}\pi_{j,i})$. The ETX has been shown to be a robust metric, especially on top of per-hop retransmissions [Gnawali et al. 2004], and is used in the MintRoute architecture [Woo et al. 2003], which treats link estimation as a complementary problem to routing. In the absence of a cap on the number of retransmissions (i.e., with unconstrained retransmissions), the ETX coincides with the RNP; it is shown in Fonseca et al. [2007] that the ETX can be measured based on the number of layer-2 acknowledgements (ACKs).

2.3 Congestion Control

Several congestion control schemes have been proposed, most of which follow a push-based paradigm whereby child nodes send their packets downstream and parent nodes request rate adaptation, either by way of hop-by-hop flow control, or through a rate-limiting scheme.

With hop-by-hop flow control, nodes signal local congestion to each other via backpressure signals, reducing packet loss rates and preventing the wasteful transmission of packets that are destined to be dropped downstream. The use of backpressure in the form of binary feedback was initially promoted as the Explicit Congestion Notification scheme in Ramakrishnan and Jain [1990]. Hop-by-hop flow control can be achieved through channel sampling [Wan et al.

[3]`http://www.tinyos.net/tinyos-1.x/tos/lib/MultiHopLQI`

2003] (estimation of channel utilization by way of periodic sampling) or queue occupancy monitoring.

Rate-limiting schemes, such as the Interference-aware Fair Rate Control (IFRC) [Rangwala et al. 2006] and Rate-Controlled Reliable Transport (RCRT) [Paek and Govindan 2007], tackle congestion control by means of rate allocation (distributed in IFRC and centralized in RCRT). In Hull et al. [2004], hop-by-hop flow control based on queue occupancy monitoring, rate-limiting, and a prioritized MAC [Woo and Culler 2001] are combined into one scheme, Fusion. Experimental evidence on a large mote network is provided, showing significant benefits. In Arbutus, we integrate hop-by-hop flow control at the network layer and study the interplay between congestion control and routing, showing that congestion control is instrumental to data collection reliability. Note that, while the interplay between congestion control and routing is not studied in Hull et al. [2004] and Rangwala et al. [2006], where routing trees are frozen, the importance of routing variability is accounted for in Paek and Govindan [2007]. The importance of congestion control for the routing performance has also been acknowledged in COSMOS (COngestion avoidance for Sensors with a MObile Sink) [Karenos and Kalogeraki 2007], where congestion control and RSS-based link estimation are jointly considered for routing to a mobile sink, and experimental results from a 10-node testbed of MICA2 nodes are presented.

A pull-based solution, the Pull Collection Protocol (PCP), has been proposed in Wachs et al. [2007]; i can only send to $p(i)$ if the latter provides a grant-to-send authorization, issued on the basis of buffer space availability. PCP eliminates the cause of congestion-induced buffer overflow at the price of a large control overhead. The rationale behind its approach is that, in the push-based paradigm, any form of congestion control is a post-facto measure, whereas the pull-based paradigm operates preemptively. Arbutus follows a push-based paradigm to avoid the considerable overhead that comes with the pull-based approach.

2.4 Load Balancing

If a routing scheme always requires nodes to choose the next-hop neighbor that offers the highest-quality route to the sink, load imbalance results in the *hot spot* problem, whereby the critical nodes experience a faster energy depletion rate due to the extra workload. Load balancing has been proposed in the form of topology control, redundancy suppression, controlled mobility, and as part of routing protocols; hybrid solutions across these categories also exist.

Redundancy suppression may be used to enhance virtually any load balancing solution (for instance, in the form of data aggregation). Siphon [Wan et al. 2007] is a hybrid between topology control and routing (cluster-based routing) that recognizes the challenge posed by the hot spot problem and proposes a multitiered solution (backed by experimental evidence on a mote testbed) based on the use of virtual sinks, special nodes equipped with a secondary radio that serve as local safety valves for overload traffic management.

Since our focus is on static homogeneous WSNs, clustering, virtual sinks, and controlled mobility are not viable options. We therefore focus on load balancing

as part of the routing protocol: Arbutus may incorporate a load-dependent term into the cost field in order to enforce parent rotation in case a particular node is overloaded [Puccinelli and Haenggi 2008a].

3. THE ARBUTUS ARCHITECTURE

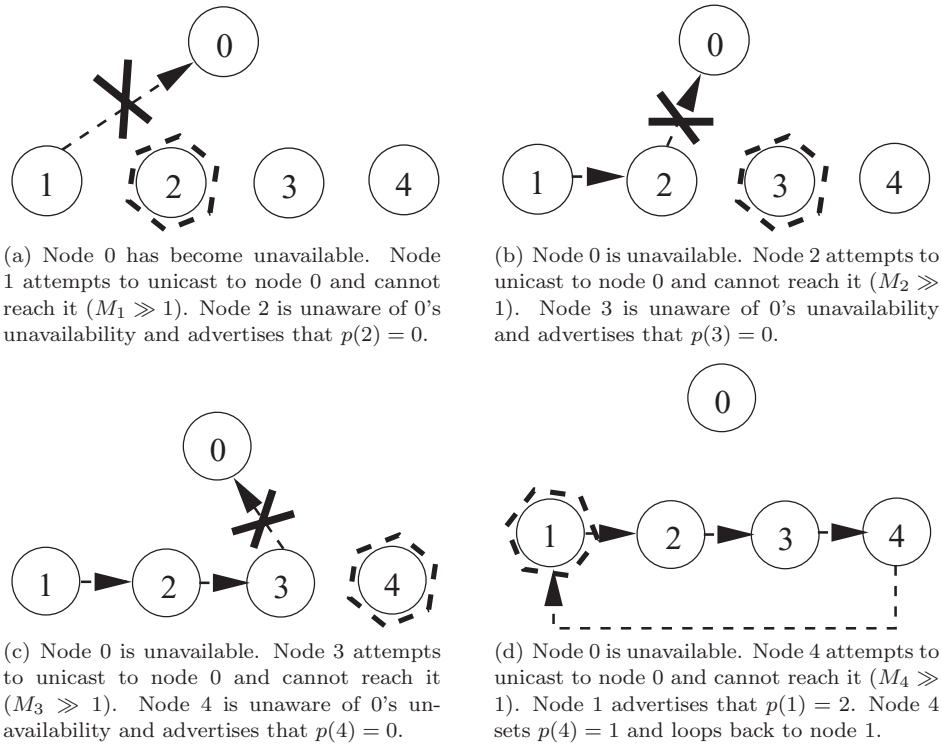
In this section we provide an in-depth description of the Arbutus routing architecture, which we break down into a data plane and a control plane. At every node, the control plane employs link estimation to build a distributed routing tree and ensure that the data plane knows the next-hop address on the way to the sink.

3.1 Data Plane

The data plane performs the basic services of data packet buffering and forwarding, complemented by duplicate packet suppression, routing loop detection, and load monitoring. The data plane also avails itself of the combined action of retransmissions and congestion control.

Basic functions. A data packet must contain three pieces of information in its header: the identity of the node that initially injected it (the packet's generator), its sequence number as assigned by the generator, and the total number of times it has been transmitted (needed to compute the RNP of a route at the sink). A unique instance of a packet is defined based on its origin address (address of its generator) and its sequence number set by its generator. The data plane unicasts all data packets to the current parent $p(i)$, which is determined by the control plane. Locally generated packets (from the application layer) and packets received from upstream neighbors are queued in a FIFO buffer. Packets are dequeued for unicast transmission to $p(i)$ only if $p(i) \neq z$. After a packet is sent to $p(i)$, if a positive layer-2 ACK is received, the FIFO buffer is served again (unless it is empty). If a layer-2 ACK is not received, the data plane implements a given retransmission policy. The data plane is also responsible for keeping track of the relayed load, which is fed into the control plane for optional network-layer load balancing.

Duplicate packet suppression. A packet duplicate may be the consequence of a failed ACK (i.e., an ACK is not sent) or a dropped ACK (i.e., an ACK is sent but not received). The latter case is more common: if i sends a data packet to j and (i, j) is asymmetric with $\pi_{i,j} > \pi_{j,i}$, j is likely to receive i's data packet, but i is likely to miss j's ACK and send the same data packet, so that j receives duplicates. Arbutus keeps duplicates from entering the data buffer by caching the signature (origin address and sequence number) of the latest packet from each node (uncapped signature caching), thus avoiding the useless and energy-inefficient propagation of duplicates at the price of extra RAM usage. Low-end nodes have a few KBs worth of RAM; given that each signature needs 2–4 bytes, uncapped caching can be used up to a network size of several hundreds of nodes.



(a) Node 0 has become unavailable. Node 1 attempts to unicast to node 0 and cannot reach it ($M_1 \gg 1$). Node 2 is unaware of 0's unavailability and advertises that $p(2) = 0$.

(b) Node 0 is unavailable. Node 2 attempts to unicast to node 0 and cannot reach it ($M_2 \gg 1$). Node 3 is unaware of 0's unavailability and advertises that $p(3) = 0$.

(c) Node 0 is unavailable. Node 3 attempts to unicast to node 0 and cannot reach it ($M_3 \gg 1$). Node 4 is unaware of 0's unavailability and advertises that $p(4) = 0$.

(d) Node 0 is unavailable. Node 4 attempts to unicast to node 0 and cannot reach it ($M_4 \gg 1$). Node 1 advertises that $p(1) = 2$. Node 4 sets $p(4) = 1$ and loops back to node 1.

Fig. 2. An example of loop formation upon route breakage: node 0 becomes unavailable and its child nodes form a loop.

Routing loop detection. Inconsistencies in the cost field may cause a node k to award parent status on an upstream peer i with $p^n(i) = k$ ($n \geq 1$), thus causing a routing loop. Trivial loops ($n = 1$) can be avoided upon parent selection using cost field information (i cannot accept k as a parent if $p(k) = i$), but loops with $n > 1$ must be specifically targeted by a dedicated scheme. Figure 2 illustrates a simple example of how a loop may arise: a node with many descendants becomes unavailable, and a time lag in the update of the state information at the descendants causes them to form a loop. In general, the formation of loops may or may not happen based on the relative timing of the control and data packets.

Arbutus has two key mechanisms that contribute to preventing the formation of loops: the outer field, whereby nodes cannot unicast data to their upstream peers, and parent pinning, whereby troubled nodes ask their child nodes to pin them. The outer field is a cost field bootstrapped by depth estimates that can be incremented if the data-driven feedback indicates that the link to the current parent is very lossy; the net result is that a node can only award parent status on its downstream peers that have a lower depth estimate, or on nodes with equal depth estimates in case its RNP blows up (for an example, refer to the description of the outer cost field in Section 3.2). Parent pinning is a data plane policy whereby a troubled node can ask its descendants to pin it as a

parent and wait out for at least a full beacon period before accepting any other node as their parent. Our extensive experiments have showed that this policy, along with the action of the outer field, is extremely effective at preventing loops from forming. Parent pinning does not cause a significant goodput loss, because it usually happens in subnetworks that have a bottleneck to the rest of the network (the troubled node is the bottleneck). If a loop does arise, the outer field can break it very easily, because it limits the depth of the nodes. While in a loop, the hop count of the nodes involved grows; as soon as a beacon is received from anywhere upstream, the beacon sender breaks the loop because its depth value is lower than all the overinflated depth values of the nodes that are stuck in the loop. As an additional safety measure, the data plane itself can detect a loop if i is forwarded a packet whose generator is i itself. Upon loop detection, duplicate suppression is disabled to avoid dropping packets in the loop, because duplicates are injected by the routing loop. The control plane is then instructed to advertise route breakage. Note that this method only detects loops that involve a packet generator (e.g., packets from node 1 in a loop 1-2-3-4-2-3-4 never travel back to 1, and so this particular loop does not involve the packet generator, node 1), but it is bound to be effective in a many-to-one traffic scenario where every node injects traffic periodically. Different traffic injection scenarios can always count on the outer field to break loops.

Retransmission strategies. Under the assumption that all packets are equally important, no particular packet should be discarded to favor others, and therefore unconstrained retransmissions are in order. If the traffic is not time sensitive, packets should be treated equally [Wachs et al. 2007]. Under the assumption of time-sensitive traffic, however, newer packets are more important than older packets, which may contain obsolete information; in this case, it makes sense to drop a packet after a given number of retransmissions $N_{\max}$. Arbutus supports both constrained and unconstrained retransmissions, but retransmissions are only performed if the node has a valid parent, or else they are put on hold. Note that the parent may of course change, and that is why Arbutus keeps track of the RNP given the current parent ($R_{i,p(i)}$) to be fair to the new parent (similarly to CTP, a new parent starts off with a clean slate and is not penalized for the shortcomings of the previous parent).

In the case of unconstrained retransmissions, the data plane retransmits as many times as necessary. In our unconstrained retransmission policy, up to $N_{\max}$ retransmissions are performed at fixed intervals T_{r}; if they do not suffice to receive a positive ACK, more retransmissions are performed at increasing intervals $T_{\mathrm{v}} = g(R_{i,p(i)})$, where $g(R_{i,p(i)})$ is an increasing function of $R_{i,p(i)}$. The reason for this strategy is that transitional links tend to be bimodal and oscillate between high and low PDR, and therefore delaying retransmissions by a fixed amount of time has been shown to reduce the transmissions-to-deliveries ratio [Srinivasan et al. 2008b]. In the case of constrained retransmissions, the data plane retransmits up to $N_{\max}$ times before dropping the packet.

In the case of unconstrained retransmissions, if more than $N_{\max}$ transmissions are needed and no alternate parent can be identified, congestion is bound to become a problem. Therefore, this particular situation is treated as the onset

of congestion (RNP-based congestion onset detection). In the reference implementation, we use $N_{\max} = 30$, $g(R_{i,p(i)}) = T_{\mathrm{r}} R_{i,p(i)}$ and $T_{\mathrm{r}} = 10$ms, and we employ unconstrained retransmissions unless otherwise specified.

Congestion control. Given that the use of unconstrained retransmissions virtually eliminates channel-related packet loss, aside from residual loss due to false ACKs or faulty CRCs, the other major factor that can cause packet loss is congestion. To mitigate it, Arbutus employs backpressure-based congestion control triggered by either an abnormal FIFO occupancy level or an abnormally lossy link to a bottleneck parent. This latter scenario, which we refer to as congestion onset detection, occurs at node i with parent $p(i) = k$ when $R_{i,k} > N_{\max}$: node i does not infer parent loss (which might be conducive to a loop), but preemptively infers congestion (because its FIFO buffer lacks an outlet).

The fraction of FIFO occupancy $q \in [0, 1]$ is fed into the state machine after each access to the FIFO buffer. The state machine has two states, a Default state and a Congested state; the Congested state is entered if $q \geq W_{\mathrm{c}}$ or in the case of congestion offset detection, and it is not exited until the queue becomes empty (to ensure hysteresis). All communication to the descendants is performed with beacons managed by the control plane according to a strategy that we describe in Section 3.2. If the medium is highly congested, these backpressure beacons may not be heard by their intended receivers, and this is exactly why a reasonably conservative calibration of the congestion thresholds is in order. As we have verified experimentally, a not-so-conservative threshold calibration (increasing the value of W_{c}) provides a slightly higher goodput but is detrimental to reliability. On the other hand, an overly conservative W_{c} causes too much control overhead and is counter-productive. We have determined $W_{\mathrm{c}} = 0.75$ to be a good compromise, as we show in Section 4.5.

3.2 Control Plane

The control plane selects the next-hop address based on a cost field set up across the network. The cost field setup is initiated by the sink and continued by the other nodes as they receive control beacons from their downstream neighbors. The control plane is built around the DoUble Cost Field HYbrid (DUCHY) link estimator [Puccinelli and Haenggi 2008b], which employs an outer field for depth control and an inner field for parent selection, allowing Arbutus to actively enforce long-hop routing while avoiding lossy links. Both cost fields are bootstrapped with beacon-based channel estimates and refined with data-driven feedback. The outer field is conducive to a low hop count (long hops), while the inner field and the data-driven feedback for both fields are instrumental to robust link estimation.

Routing state. Table II shows the routing state variables and indicates which ones are advertised in the broadcast control beacons. At a given node i, Arbutus only keeps state for the current parent $p(i)$ and keeps track of the former parent $\tilde{p}(i)$. The address of the current parent is advertised in the control beacons and is denoted as $p_a(i)$. Two kinds of beacons are employed: route beacons advertising a valid route ($p_a(i) = p(i) \neq z$), which are broadcast at

Table II. State Information at Node i

State Variable	Symbol	In Beacon
address	i	yes
current parent address	$p(i)$	no
former parent address	$\tilde{p}(i)$	no
advertised parent address	$p_a(i)$	yes
hop distance to sink	H_i	yes
bottleneck link RSS	Λ_i^{nRSS}	yes
bottleneck link LQI	Λ_i^{nLQI}	yes
bottleneck relayed load	B_i	yes
local relayed load estimate	β_i	no
cost of reaching the sink by way of $p(i)$	C_i	no

For each state variable, we provide the corresponding symbol and specify whether it is included in the control beacons.

regular intervals (every 60s in the implementation), and no-route beacons advertising route breakage ($p_a(i) = z$), which are broadcast whenever necessary. Route breakage at node i is defined as the lack of a valid parent ($p(i) = z$); it is advertised veridically ($p_a(i) = p(i) = z$) in the case of a routing loop or downstream congestion and is faked ($p_a(i) = z \neq p(i)$) in the case of local congestion. Inferring route breakage based on data plane feedback (i.e., in case the RNP blows up) is not necessary, because the double cost field can mend itself if other parents are available. We have also observed that inferring route breakage based on data plane feedback may be conducive to routing loops.

Double cost field link estimator. The centerpiece of the control plane is the DUCHY link estimator [Puccinelli and Haenggi 2008b]. Any beacon from node b advertising $p(b) \neq z$ is subjected to a vetting process by the double cost field. The two cost fields are bootstrapped with CSI and refined with data-driven feedback, the RNP, which is measured by counting layer-2 ACKs (as in Fonseca et al. [2007]). Specifically, the outer field is initialized with depth estimates refined with the RNP, while the inner field is bootstrapped with CSI based on the received control beacons and also refined with the RNP. DUCHY is hybrid in two different ways: it is driven by both CSI and RNP estimates, and it exploits both broadcast control traffic and data traffic (like Woo et al. [2003] and Fonseca et al. [2007]).

Data-driven feedback is essential because the control plane cannot take beacon-based link estimates at face value [Zhang et al. 2008]: for one, CSI is based on reverse-link estimates (beacons travel over $(p(i), i)$, but data will travel over $(i, p(i))$), and the timescale of data traffic is necessarily different from the timescale of control traffic. Specifically, it has been observed [Srinivasan et al. 2008a] that the PDR over a given link may be different for traffic with different inter-packet intervals. Further, since the quality of wireless links may change over time (for instance, due to shadowing and induced fading [Puccinelli and Haenggi 2006b]), beacon-based link estimates may provide obsolete information due to the propagation lag of the cost field, especially in the case of transitional links (because very good links typically have a large noise margin).

For platform independence, DUCHY could obtain CSI from PDR estimates based on control beacon sequence numbers, but link-layer ACKs would still be required for data-driven feedback, as is the case with Four-Bit Link Estimation [Fonseca et al. 2007]. Since 802.15.4 provides link-layer ACKs as well as the LQI, and since the RSS is available with most radios, our implementation of DUCHY employs both the RSS and the LQI. The RSS is used to obtain soft information about good links, while the LQI is used to get soft information about bad links. The CSI-based estimate obtained by merging the RSS and the LQI is refined through the use of RNP feedback from the data plane. The version of DUCHY used in this article, differently from the original version in Puccinelli and Haenggi [2008b], allows the option of network-layer load balancing by incorporating load information into the inner field.

Outer cost field. The purpose of the outer cost field is to limit the depth of the routing tree. At node i, the outer field filters out neighbors that are estimated not to be closer to s than i (i.e., nodes estimated to be at the same depth as i, or deeper in the tree), because they would lead to unnecessarily long routes. Let H_i be the hop count between i and s. The outer cost field actively pursues long-hop routing, and the outer cost normally coincides with the hop count, so that nodes can only award parent status to shallower nodes. The hop count, however, is a beacon-based estimate, and taking beacons at face value for the estimation of a node's depth in the routing tree can easily cause nodes to underestimate their depth. Counting hops, in fact, means assuming links to be Boolean objects that either exist or do not exist, but low-power wireless links are probabilistic. We therefore refine our beacon-based depth estimates with data-driven feedback, which we distil down to a binary depth correction term c that is normally equal to 0 and is set to 1 if the current parent of i is deemed to be particularly unreliable ($R_{i,p(i)} \gg 1$). The outer cost is given by

$$A_i \triangleq H_i + c, \tag{1}$$

and a beacon from node b is said to pass through the outer cost field at i if $A_b < A_i$. Computing the outer cost as an incremented hop count enables beacons from a prospective parent b that advertises $H_b = H_i$ to pass through the outer field in case i is stuck with a lossy link. The exact RNP threshold beyond which a parent is tagged as unreliable and c is set to 1 is application-dependent (we set it to 5 in the reference implementation).

Data-driven feedback is essential to refine beacon-based estimates and avoid link estimation errors. While beacon-based link estimates only give information on inbound links, data plane feedback makes link estimates bidirectional, so that asymmetric links can be avoided. Suppose, for instance, that i estimates $H_i = 2$ and $p(i) = k$. Further, suppose that (k, i) is asymmetric with $\pi_{k,i} > \pi_{i,k}$: i generally receives k's beacons, but k generally misses i's data packets. The outer cost is bootstrapped as $A_i = 2$, i starts using (i, k), but the RNP blows up: $R_{i,p(i)} \gg 1$. Suppose now that all the other neighbors of i also lie at depth 2. Without data-driven feedback, i would be stuck with $p(i) = k$. Since $R_{i,p(i)} \gg 1$, however, its outer cost becomes $A_i = 3$, and the beacons from i's neighbors k with $H_k < 3$ are allowed through the outer field, which means that i can use the inner cost field to choose a new parent among them.

The outer field is very helpful with routing loops. As an example, in Figure 2(c), $A_4 = 1$ ($H_4 = 1$ and $c = 0$), $A_3 = 2$ ($H_3 = 1$ and $c = 1$, because $M_3 \gg 1$), $A_2 = 3$ ($A_2 = A_3 + 1$), and $A_1 = 4$. Therefore, in Figure 2(d), node 4 cannot award parent status on any of its peers, because $A_4 < A_k$ with $k = 1..3$.

Inner cost field. While the outer field directs the choice of a prospective parent toward the set of downstream neighbors, the inner field is in charge of awarding parent status on a specific downstream neighbor. If a beacon passes through the outer field, the link estimator obtains the necessary CSI measurements related to the received beacon as feedback from the lower layers and marshals the relevant arguments for the computation of the inner cost of choosing b, the beacon's sender, as its new parent. As node i considers node b as a prospective parent, it needs to pay attention to the outgoing link (i, b), which is the first step out to s: a significant quality fluctuation on (i, b) would disrupt communication over $[i, s]_b$. Of course, it is just as important for i to avoid that packets get stuck at some point past b. Due to the different timescales of control and data traffic and the time lag between the beacon arrival and the actual use of the corresponding link, the reliability of beacon-based estimates is relatively low. Therefore, rather than keeping track of the quality of all links in a route (e.g., with an additive link metric), we only account for the worst link (bottleneck link) of a given route (in addition to the outgoing link).

The inner cost C_i^b of using b to reach s, that is, the cost of using $[i, s]_b$, is computed by i as a function of both the quality of (b, i) and the quality of the worst link (link quality bottleneck) over $[i, s]_b$:

$$C_i^b = \mathrm{nCSI}_{b,i} + \Lambda_b^{\mathrm{nCSI}} + w_{\mathrm{B}} B_b, \tag{2}$$

where $\mathrm{nCSI}_{b,i}$ denotes normalized CSI measured at i over (b, i), $\Lambda_b^{\mathrm{nCSI}}$ represents the bottleneck normalized CSI advertised by b, defined as

$$\Lambda_b^{\mathrm{nCSI}} \triangleq \max_{r \in [b,s]_{p(b)}} \mathrm{nCSI}_{p(r),r}, \tag{3}$$

B_b represents the load bottleneck seen by b, defined as

$$B_b \triangleq \max_{r \in [b,s]_{p(b)}} \beta_r, \tag{4}$$

and w_{B} is a binary selection variable that allows us to select or deselect the option of load balancing. The load bottleneck term embeds load balancing into the routing protocol, because it raises the cost of a prospective parent with a large relayed load (even if it advertises a reliable route). If $w_{\mathrm{B}} = 1$, we speak of Arbutus with load balancing, while we speak of Arbutus without load balancing if $w_{\mathrm{B}} = 0$. The values of $\Lambda_b^{\mathrm{nCSI}}$ and B_b are advertised in the control beacons broadcast by b to ensure state propagation and build up a cost field.

Our reference implementation takes full advantage of the two kinds of CSI provided by the CC2420: the RSS and the LQI. We denote the normalized RSS over (b, i) as $\mathrm{nRSS}_{b,i}$ and the normalized LQI as $\mathrm{nLQI}_{b,i}$, and introduce binary selection variables w_{RSS} and w_{LQI} to select and deselect each piece of CSI in our evaluation. RSS is normalized so that 0 corresponds to a very high RSS

(> -50dBm), and 1 corresponds to a very poor RSS (< -85dBm), and LQI is normalized so that 0 corresponds to a very high LQI (> 100), and 1 corresponds to a very poor LQI (< 60). Unless otherwise noted, $w_{\mathrm{RSS}} = 1$ and $w_{\mathrm{LQI}} = 1$. In the implementation, (2) is used in the form

$$C_i^b = w_{\mathrm{RSS}}\left(\mathrm{nRSS}_{b,i} + \Lambda_b^{\mathrm{nRSS}}\right) + w_{\mathrm{LQI}}\left(\mathrm{nLQI}_{b,i} + \Lambda_b^{\mathrm{nLQI}}\right) + w_B B_b, \tag{5}$$

where the various terms with the RSS and LQI subscripts have the same role of the corresponding terms with the CSI subscript described earlier. If b is the current parent ($p(i) = b$), then we use the simplified notation C_i for the inner cost field at i.

When a beacon received from another node, say j, passes through the outer cost field (i.e., $A_j < A_i$), node j is awarded parent status if $C_i^j < C_i + cR_{i,p(i)}$, where $cR_{i,p(i)}$ is a data-driven feedback term that artificially inflates the inner cost of the current parent if $c = 1$. Recall that c is the binary depth correction term that favors switching to a reliable prospective parent in case the current parent is found to be unreliable.

State management upon parent update. A key point is that no routing table is employed: state is only maintained for the current parent $p(i)$. A table-free protocol uses less RAM, and, much more significantly, scales very well. Aside from the current parent $p(i)$, the control plane also keeps track of the former parent $\tilde{p}(i)$, which is used to streamline congestion recovery. Changes in the state variables after the adoption of a new parent include the update of the load, the RSS, and the LQI downstream bottlenecks, the update of the hop count to the sink ($H_i \triangleq H_{p(i)} + 1$, which impacts the outer cost A_i), and the update of the inner cost C_i.

Recovery from abnormal conditions. If a parent becomes unavailable for any reason, its child nodes no longer receive its beacons. Rather than using an arbitrary timeout value to infer parent loss, we let the cost fields mend themselves (if possible): the data plane keeps unicasting to $p(i)$, the RNP blows up, so does A_i, and as soon as a beacon from a different neighbor j with $A_j \leq A_i$ is received, j is awarded parent status. As we have already mentioned, a large RNP to $p(i)$ is viewed by i as the onset of congestion (RNP-based congestion onset detection), and a no-route beacon is broadcast to keep the descendants from sending in more data packets in case it is not possible to mend the cost field.

In the case of local congestion at node j, the congestion control state machine in the data plane enters the Congested state, where node j requires its descendants to stop sending by faking parent loss. Specifically, node j broadcasts a no-route beacon with $p_a(j) = z$: j gets its descendants to stop sending data packets by faking route breakage ($p(j) \neq z$, but $p_a(j) = z$).

Routing loops and downstream congestion are both viewed as a route breakage. If $p(i) = k$ and node i receives a packet generated at i, a loop is detected, which automatically means that k does not have a valid route to the s. If $p(i) = k$ and k broadcasts a no-route beacon with $p_a(k) = z$, i learns that its current parent k is congested; for i's purposes, k cannot currently offer a valid route to s.

Route breakage is normally conducive to a high risk of routing loop formation. To prevent loops from forming, Arbutus employs *parent pinning*: upon reception of a no-route beacon from its parent k, i pins the identity of k and does not switch to a new parent for at least one full beacon period. Pinned parents are blacklisted and only chosen again as parents if there are no viable alternatives, in which case the formerly pinned parent is bound to be a bottleneck and is taken off the black list.

4. EXPERIMENTAL PERFORMANCE ANALYSIS

4.1 Methodology

Evaluation on testbeds. We perform a comprehensive routing evaluation and benchmarking on large-scale testbeds. For many different reasons, the number of working nodes in a given testbed fluctuates over time: hardware may fail, software issues may occur, and human activity may disturb the nodes. As a reference, we provide the number of working nodes in each testbed that we use averaged over the timeframe of usage. Most of the experimental work for this article was carried out on the MoteLab testbed over several months. All the data presented herein pertains to three particular timeframes of testbed usage: March 2008, when the average number of working nodes was 90, April–May 2008, when the average number was 155, and December 2008, when the average number was 130. An additional set of results obtained in June 2009 with an average of 100 nodes is also presented. MoteLab nodes are distributed over 3 floors. The total deployment area is about 6500 square meters, which corresponds to a relatively low average density of approximately 0.025 nodes per square meter. Node placement is very irregular, and the node degree has a very high variance. We also present results obtained on the Telecommunication Networks Group Wireless Indoor Sensor network Testbed (Twist) at the Technische Universität Berlin, and on the Tutornet testbed at the University of Southern California. In terms of node placement regularity, Twist is at the other end of the spectrum with respect to MoteLab, as its 102 nodes are arranged in a fairly regular grid over 3 floors; each floor covers an area of 460 square meters, and the average density is a relatively high 0.074 square meters, about 3 times the density of MoteLab (there are 204 sockets for 102 TMoteSky and 102 eyesIFX nodes; in the experiments in this article, we employ the 102 TMoteSky nodes). Tutornet contains 91 TMoteSky nodes distributed over sections of 2 floors of 1400 square meters each, with an average density of about 0.065 nodes per square meter. Using all three testbeds provides us with topological diversity.

CTP as a benchmark. Arbutus lies in the same subset of the design space as the MintRoute family, so CTP represents a natural choice for a benchmark. The control plane of CTP sets up a cost field based on link estimation. CTP's control plane treats the link estimator as a separate block, so that CTP may use different link estimators; in this work, we employ both the Link Estimation Exchange Protocol (LEEP) [Gnawali 2007] and Four-Bit Link Estimation

[Fonseca et al. 2007]. In LEEP, a neighbor table is seeded with bidirectional ETX estimates based on control beacons (specifically, we employ the implementation in `http://www.tinyos.net/tinyos-2.x/tos/lib/net/le`). With Four-Bit Link Estimation, LEEP's beacon-based estimates are refined on the basis of data plane feedback (link-layer ACKs are used to measure the RNP). While LEEP was the standard link estimator when we began this work, Four-Bit has now taken over, which is why we employ it in the more recent experiments. Arbutus's link estimator is tightly integrated into the control plane, so in this sense Arbutus is not as modular as CTP. Arbutus's DUCHY-like link estimator uses both CSI (RSS and LQI) and ETX-like quantities. CTP also uses these pieces of information, but at the lower granularity of one bit (the *white bit* in the Four-Bit Link Estimator). Like Arbutus, CTP also sets up its cost field through beaconing; CTP's interbeacon intervals, however, are not constant, but are adapted. CTP does not perform load balancing. CTP's data plane provides buffering, loop detection, and congestion detection, but no congestion control, other than rate-limiting. There are two levels of buffering: a message pool and a forwarding queue. The idea behind the message pool concept is that multiple network protocols might coexist and need to exchange information with the link layer. While the message pool concept is certainly applicable to Arbutus, we limit ourselves to using a forwarding queue of the same size as CTP's (12 packets).

CTP has built-in loop detection and breakage features. A data packet advertises its sender's cost of reaching the sink, and if a node receives a data packet advertising a cost no greater than its own, it infers that there must be an inconsistency in the cost field. CTP then broadcasts a control beacon to inform the lower-cost descendant that the cost field has become inconsistent. If the lower-cost descendant does not hear the beacon, the loop does not get broken and the cost of the nodes involved keeps increasing, but CTP breaks the loop when the cost value exceeds an implementation-dependent threshold. Differently from CTP, Arbutus uses several loop prevention techniques, as described in Section 3.2; if loops do arise, Arbutus breaks out of them thanks to the depth-limiting outer field itself, along with a simple loop detection scheme (see Section 3.1).

Another difference lies in duplicate suppression: CTP caches a finite number of signatures (made up of address, sequence number, and a Time Has Lived field), while Arbutus uses a reduced signature (address and sequence number), but stores it for the latest packet received from all nodes. Arbutus's memory signature (4 bytes per node) therefore grows linearly with the number of nodes, but it is relatively small for the network size that we consider (100–150 nodes). In much larger networks (e.g., more than 1000 nodes), Arbutus would need to employ CTP's limited-size signature caching approach.

While Arbutus has the option of unconstrained retransmissions, CTP gives up on a packet after 30 retransmissions. Arbutus's unconstrained retransmissions require a backpressure scheme for congestion control. In CTP, aside from a rate-limiting mechanism, congestion control is not used, but congestion detection is employed, which means that CTP facilitates the use of

existing congestion control schemes without attempting to incorporate their functionality.

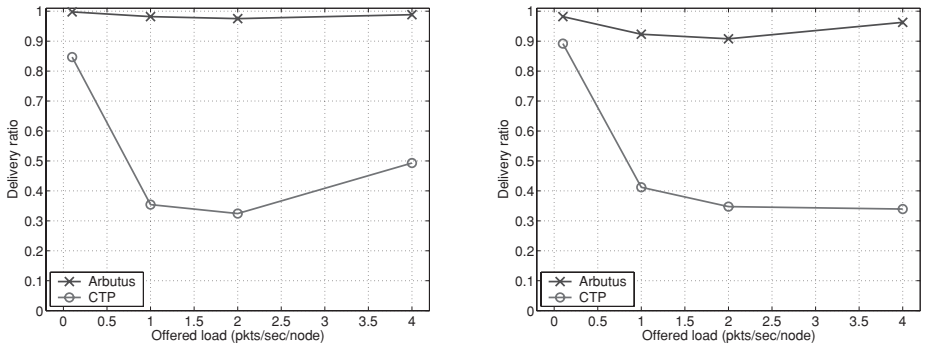
4.2 Experimental Setting and Performance Metrics

We use a many-to-one application scenario where each node generates traffic at a fixed target rate f_{gen} (the target offered load, equal for all nodes) destined to a single sink node. It is assumed that all nodes are equal and all packets have the same importance: there is no benefit in dropping a given packet to favor another one. Due to the presence of a congestion control mechanism, the actual offered load may be lower than the target offered load. In general, each experiment has two main parameters: the sink assignment and the target offered load. For each MoteLab data point, we typically present results obtained as an average over 3 experimental runs of a duration ranging from 10–30 minutes (most runs last 15 minutes). Experiments with Arbutus and the benchmark, CTP, are run back-to-back. Network-wide results are obtained by averaging over all nodes. For Twist, we typically present results obtained as an average over 1–2 runs of an average 15–20 minutes. For Tutornet, we show fewer but longer experiments, typically 1 run per data point, ranging from 30 minutes to a few hours. For the March 2008 and the April–May 2008 MoteLab experiments, as well as for the Twist and Tutornet experiments, we employ the February 2008 release of CTP with LEEP. For the MoteLab experiments from December 2008 and later, we use the September 2008 release of CTP with Four-Bit Link Estimation. We use a transmit power of 0dBm for every node in all experiments. We employ several performance metrics for our experimental evaluation.

Delivery ratio. Since Arbutus is optimized for reliability, the primary performance indicator is the delivery ratio, whose average for the network is computed as the total number of delivered packets over the total number of sent packets.

Goodput. It is defined as the number of successfully delivered packets at the sink per time unit (we measure it in packets per second). Delivery ratio and goodput provide nonoverlapping information: a protocol can achieve a relatively large goodput at the cost of a low average delivery rate (for instance, by adopting a best-effort approach and injecting more packets than can be accommodated), or, conversely, it can achieve a very high delivery rate at the cost of a goodput degradation and a larger delay (for example, through the use of unconstrained transmissions).

Coverage. We define coverage with respect to target values, which are fractions ρ of the offered load. A node is said to be covered with respect to a target value ρ if its contribution to the goodput at the sink is no less than a fraction ρ of its offered load. As an example, if $\rho = 0.75$ and $f_{\text{gen}} = 1$ pkt/sec/node, then a node is covered with respect to ρ if it is able to contribute a goodput of no less than 0.75 pkts/sec at the sink. Note that this notion of coverage is agnostic to the delivery ratio, because what matters is only the number of packets delivered to the sink per time unit. In other words, if $\rho = 0.75$ and $f_{\text{gen}} = 1$



(a) MoteLab, 90 nodes (b) MoteLab, 155 nodes

Fig. 3. Delivery ratio with Arbutus (with load balancing) and CTP (with LEEP) at four different levels of offered load for two different network sizes.

pkt/sec/node, a node that delivers 0.8 pkts/sec and loses 0.2 pkts/sec is considered to be covered, while a node that delivers 0.7 pkts/sec with no packet loss (thanks to the use of retransmissions) is considered not to be covered.

Fairness. Fairness is a widely used metric in the congestion control literature [Hull et al. 2004]. It is usually applied to delivery rates according to Jain's equation [Jain 1991]:

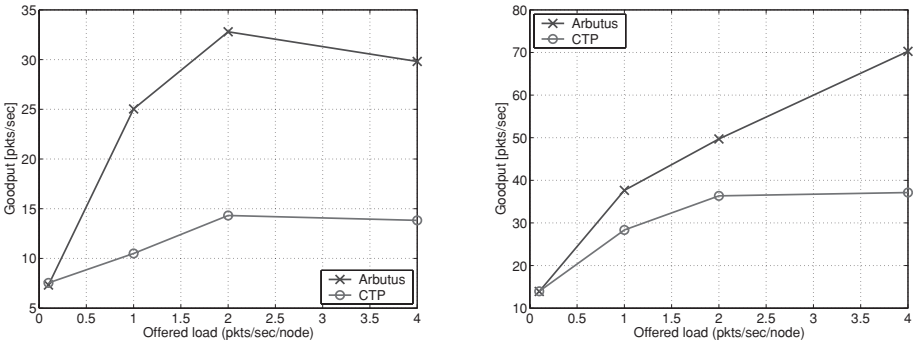
$$\phi \triangleq \frac{\left(\sum_{i\in\mathcal{N}} e_i\right)^2}{|\mathcal{N}| \sum_{i\in\mathcal{N}} e_i^2}, \tag{6}$$

where e_i is the delivery ratio of node i measured at s. While coverage is agnostic to the delivery ratio and only considers the goodput, fairness is agnostic to the goodput and only considers delivery ratios.

Routing cost. The routing cost from a node i to s is defined as the total number of transmissions needed to deliver a packet over $[i, s]$. The routing cost for a network $\mathcal{N}$ is computed as the average of the routing costs from the nodes $i \in \mathcal{N}$. Ideally, the routing cost should coincide with the average hop distance to the sink, but this is not the case due to the lossy nature of wireless links and the consequent need for retransmissions.

4.3 Performance as a Function of the Offered Load

We begin by assessing the impact of the offered load on Arbutus and CTP. We use Arbutus with load balancing ($w_B = 1$ in Eq. (5)) and CTP with LEEP. We arbitrarily fix the sink assignment (we choose MoteLab's node 151) and vary the offered load. For a given value of the offered load, we plot the delivery ratio and goodput averaged over 5 experiments that we ran within a few hours of each other. We consider four different offered load levels: $f_{\text{gen}} = 0.1$ pkt/sec, $f_{\text{gen}} = 1$ pkt/sec, $f_{\text{gen}} = 2$ pkts/sec, and $f_{\text{gen}} = 4$ pkts/sec. Figures 3(a) and 3(b) show the delivery ratio in, respectively, the 90-node from March 2008 and the 155-node testbed from April–May 2008.



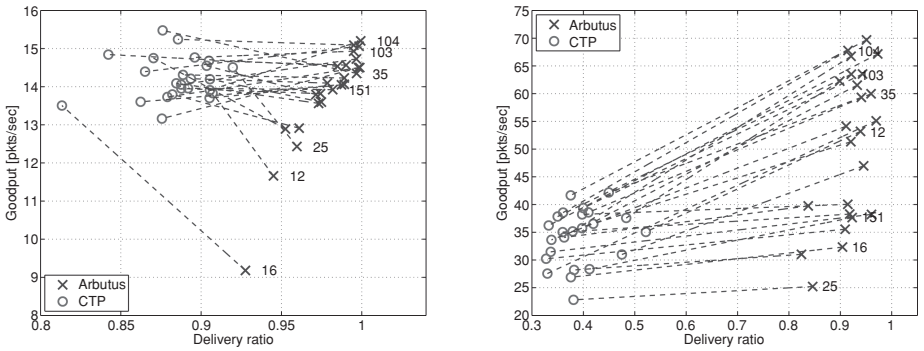
(a) MoteLab, 90 nodes (b) MoteLab, 155 nodes

Fig. 4. Goodput with Arbutus (with load balancing) and CTP (with LEEP) at four different levels of offered load for two different network sizes.

In terms of reliability (Figure 3) Arbutus significantly outperforms CTP at all the offered load points, and its performance is relatively insensitive to the variations in the offered load. Figures 4(a) and 4(b) show the goodput achieved by Arbutus and CTP in, respectively, the 90-node and the 155-node testbed. At 0.1 pkt/sec, CTP slightly outperforms Arbutus in terms of goodput in the 90-node network and matches Arbutus's performance in the 155-node network, while Arbutus does much better at higher offered loads. If the offered load is low, Arbutus's load balancing definitely affects the goodput, while at higher offered load points, the interplay between load balancing and congestion control kicks in. Load balancing keeps nodes from getting overloaded and prevents congestion: under heavy load, the goodput does not suffer as much as under light load.

Figure 4(a) shows that, with Arbutus, the goodput in the 90-mote network peaks at $f_{\text{gen}} = 2$ pkts/sec; this does not happen in the 155-mote network, where Arbutus's goodput increases monotonically with the offered load (Figure 4(b)). This is due to the presence of bottlenecks in the 90-mote network: these bottleneck nodes get congested and thus limit the achievable goodput as the target offered load increases. In the 155-mote network, congestion is not as critical: there are no significant bottlenecks (because the bottlenecks in the 90-mote network were due to the unavailability of 65 motes that are now available), and the achievable goodput at $f_{\text{gen}} = 4$ pkts/sec is therefore larger. CTP's lack of congestion control (other than rate-throttling) is responsible for the saturation of its goodput past $f_{\text{gen}} = 2$ pkts/sec at levels well below Arbutus's in both versions of the MoteLab testbed. As CTP specifically targets relatively low-rate data delivery, it does not contain an explicit congestion control mechanism. Arbutus, on the other hand, contains several congestion control mechanisms to accommodate a relatively high offered load without a significant performance degradation.

In the remainder of the article, we will concentrate on only two offered load points: $f_{\text{gen}} = 0.1$ pkt/sec and $f_{\text{gen}} = 1$ pkt/sec, because the most remarkable performance changes occur in this interval. Given that we work with a network



(a) $f_{\text{gen}} = 0.1$ pkts/sec (b) $f_{\text{gen}} = 1$ pkts/sec

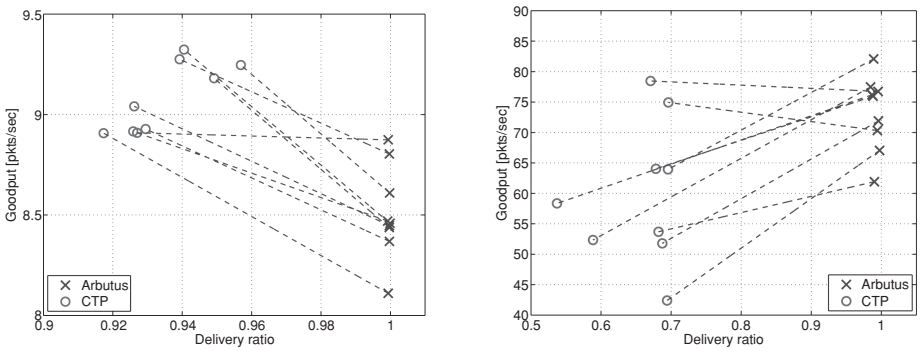
Fig. 5. Performance plane for Arbutus (with load balancing) and CTP (with LEEP) in MoteLab (about 155 nodes). Results at several sink assignments are shown, and in a few cases the sink assignment itself is indicated. Different sink assignments correspond to different network topologies. The impact of topology on the goodput is significant, particularly at higher offered load points (note the different scale of the two subfigures). Arbutus's delivery ratio is, however, relatively insensitive to the topology.

size between 90 and 160 nodes, the offered load point $f_{\text{gen}} = 0.1$ pkt/sec will be henceforth referred to as *light load*, while $f_{\text{gen}} = 1$ pkt/sec will be considered *heavy load*.

4.4 Joint Impact of the Topology and the Offered Load

Topology as a parameter. The results in Section 4.3 pertain to a specific, arbitrary sink assignment in the MoteLab testbed. In the assessment of a routing protocol, network topology must be treated as a parameter, similarly to the target offered load. To understand how crucial topology is, we perform several experiments with different sink assignments. We continue to use Arbutus with load balancing ($w_{\text{B}} = 1$) and CTP with LEEP, and we distill the performance indicators down to their average value over all nodes in the network, after obtaining each per-node value as the average over at least 3 runs. In particular, we visualize the routing performance on a goodput-delivery ratio plane (henceforth called *performance plane*), and we represent the routing cost on a hop count-total number of transmissions plane (henceforth named *cost plane*).

Performance plane. Figure 5 shows the performance plane for a number of network topologies in MoteLab. We observe that the goodput varies significantly depending on the sink assignment, especially for Arbutus. This happens because Arbutus's long-hop approach is able to leverage on the network topology, and a change in the sink assignment means a different topology. In general, CTP yields a better goodput under light load; this is particularly noticeable in topologies 12, 25, and, in particular, 16, which all appear to be performance outliers. Load balancing causes a definite goodput degradation under light load; this degradation may be particularly significant depending on the topology.



(a) $f_{\text{gen}} = 0.1$ pkts/sec (b) $f_{\text{gen}} = 1$ pkts/sec

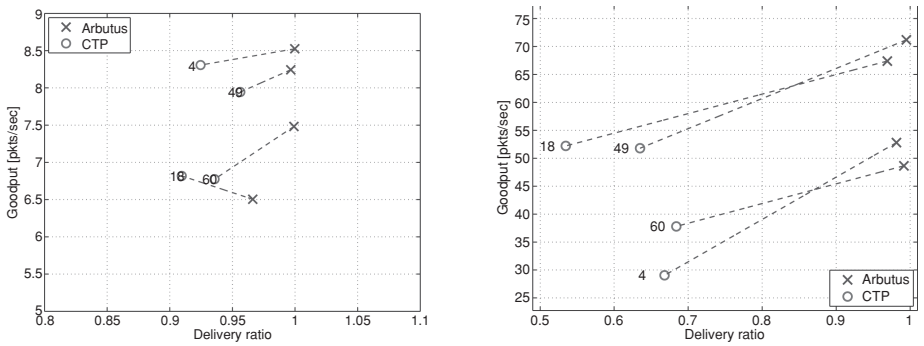
Fig. 6. Performance plane for Arbutus (with load balancing) and CTP (with LEEP) in Twist (about 100 nodes). The impact of topology on the goodput is not as significant as in MoteLab, given the regular grid-like structure of the Twist network.

Figure 5 also shows that the delivery ratio of Arbutus is always superior and relatively insensitive to both topology and load. Arbutus owes this to its long-hop approach and the combined action of congestion control and unconstrained retransmissions. CTP's delivery ratio also appears to be insensitive to the sink assignment; we conclude that topology does not have a significant impact on the delivery ratio.

Figure 6 shows the performance plane for a few sink assignments in Twist. Due to its very regular topology, there is very little variability across different sink assignments, and the performance of both protocols is considerably better than in MoteLab. Arbutus achieves virtually 100% reliability under light load, and is always above 98% reliability under heavy load. Twist's regular grid-like topology is not conducive to congestion, and CTP's goodput is consistently better than Arbutus's under light load (Figure 6(a)). While load balancing hurts the goodput under light load, the interplay between congestion control and load balancing begins to kick in under heavy load (Figure 6(b)), and Arbutus nearly always outperforms CTP in terms of goodput. Even under high load, CTP's goodput is better than Arbutus's for a couple of sink assignments, for which Twist's regular topology does not trigger the interplay between load balancing and congestion control (because the topology is so regular that there is hardly any congestion).

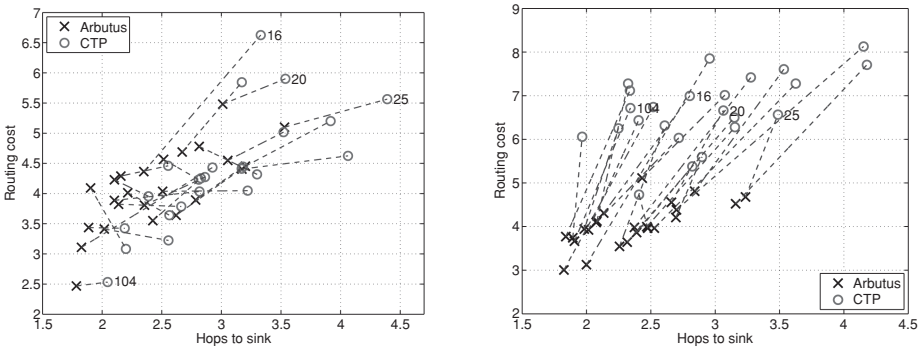
Figure 7 shows the results on Tutornet; in particular, Figure 7(a) shows that on Tutornet Arbutus (with load balancing) achieves a competitive goodput performance even under light load, due to the irregular distribution of the nodes that is conducive to congestion. Tutornet also suffers from a heavy amount of 802.11 interference; although these results were obtained on a nonoverlapping 802.15.4 channel (channel 26), there remains a certain amount of residual interference that increases the average RNP, contributing to the injected load.

Cost plane. In MoteLab, the routing cost, shown in Figure 8, is affected by wide performance variations depending on the sink assignment. Arbutus



(a) $f_{\text{gen}} = 0.1$ pkts/sec (b) $f_{\text{gen}} = 1$ pkts/sec

Fig. 7. Performance plane for Arbutus (with load balancing) and CTP (with LEEP) in Tutornet (about 90 nodes). Topology has an impact on the goodput, albeit not as much as in MoteLab, due to Tutornet's higher density.

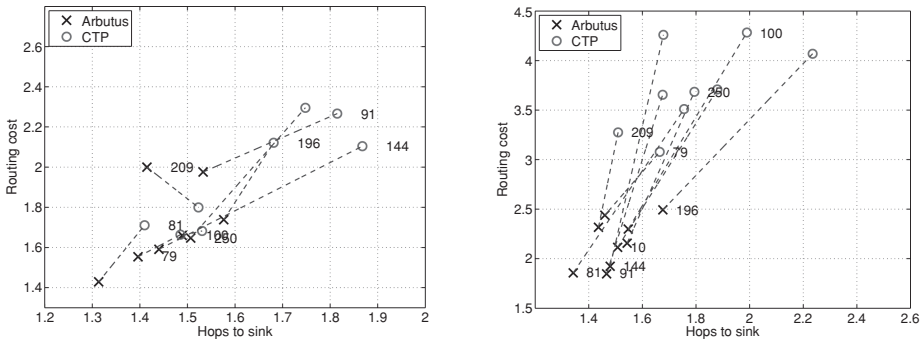


(a) $f_{\text{gen}} = 0.1$ pkts/sec (b) $f_{\text{gen}} = 1$ pkts/sec

Fig. 8. Cost plane for Arbutus (with load balancing) and CTP (with LEEP) in MoteLab, 155 nodes. Arbutus greatly reduces the routing cost (transmissions per delivered packet) as well as the depth of the routing tree. Note the different scale in the two subfigures, and the relative consistency of Arbutus's routing cost as the offered load is increased.

typically achieves a smaller routing cost with fewer transmissions per delivered packet. Its routing cost is relatively insensitive to the offered load, while CTP's cost varies greatly between the two offered load points. This is due to Arbutus's more efficient tree structure: unnecessarily long routes are avoided, and a smaller relayed load is imposed onto the network.

The cost plane for our Twist experiments is shown in Figure 9. Given the regular grid-like layout of Twist, Arbutus's average routing cost is fairly consistent across different sink assignments and always lower than CTP's. Like on MoteLab, Arbutus's cost does not vary significantly as the target offered load is increased. Moreover, Arbutus always builds shallower trees, thanks to its long-hop routing strategy.



(a) $f_{\text{gen}} = 0.1$ pkts/sec (b) $f_{\text{gen}} = 1$ pkts/sec

Fig. 9. Cost plane for Arbutus (with load balancing) and CTP (with LEEP) in Twist, about 95 nodes. Due to the regular grid-like layout of the network, all sink assignments correspond to benign topologies, and the performance is consistent across different topologies.

Table III. MoteLab, 155 nodes: Overall Results for the April–May 2008 Experiments

Performance Indicator	Arbutus (light)	CTP (light)	Arbutus (heavy)	CTP (heavy)
Delivery ratio	0.98 ± 0.02	0.89 ± 0.02	0.92 ± 0.03	0.40 ± 0.06
Goodput [pkts/sec]	13.95 ± 1.09	14.26 ± 0.31	52.70 ± 13.95	34.10 ± 4.65
Hop count	2.65 ± 0.50	3.22 ± 0.56	2.45 ± 0.41	3.04 ± 0.61
Routing cost	4.25 ± 0.68	5.72 ± 0.74	4.05 ± 0.48	7.67 ± 0.90
Coverage at 5%	0.99 ± 0.01	0.99 ± 0.01	0.78 ± 0.12	0.77 ± 0.11
Coverage at 25%	0.99 ± 0.01	0.99 ± 0.01	0.50 ± 0.16	0.32 ± 0.09
Coverage at 50%	0.98 ± 0.01	0.99 ± 0.01	0.27 ± 0.10	0.15 ± 0.05
Control overhead	0.17 ± 0.03	0.09 ± 0.05	0.05 ± 0.02	0.26 ± 0.20
Duplicate suppression	0.99 ± 0.01	0.94 ± 0.03	0.99 ± 0.01	0.85 ± 0.04

We compare Arbutus with load balancing and CTP with LEEP, both under light and heavy load. For each performance indicator, we provide the mean μ and the standard deviation σ in the form $\mu \pm \sigma$. The results are averaged over all the topologies that we considered.

Overall benchmarking results. In Table III we present an overview of our April-May 2008 experimental results on the 155-node MoteLab testbed for the two offered load points of 0.1 and 1 pkt/sec/node, averaged over all topologies. Arbutus definitely achieves reliability, as proved by the 10% improvement over CTP under light load and the wide gap under heavy load. Load balancing does degrade the goodput under light load: Arbutus falls 3% below CTP's goodput. Due to the interplay between load balancing and congestion control, however, Arbutus outperforms CTP by as much as 50% under heavy load. Independently of the offered load, Arbutus reduces the average node depth by an average 19%, and reduces the number of transmissions per delivered packet by 47% under light load, and by 26% under heavy load. The coverage of both protocols is virtually identical under light load. Under heavy load, however, Arbutus outperforms CTP. Under light load, Arbutus's control overhead is almost double than CTP. There are two reasons for this: Arbutus's congestion control scheme, and the fact that CTP relaxes the beaconing interval if no anomalies are detected. The situation is completely reversed under heavy load: Arbutus's control

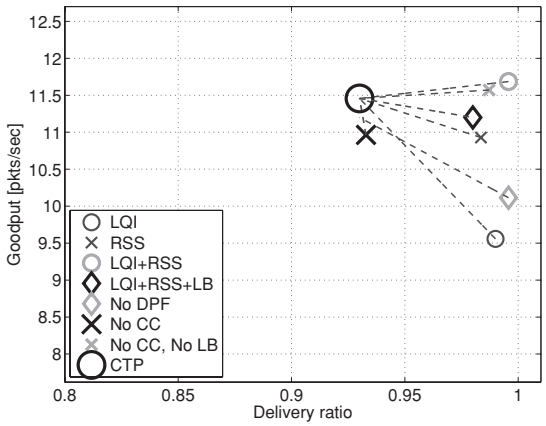


Fig. 10. MoteLab, 130 nodes (December 2008). Relative impact of the components of the Arbutus architecture on the performance plane under light load.

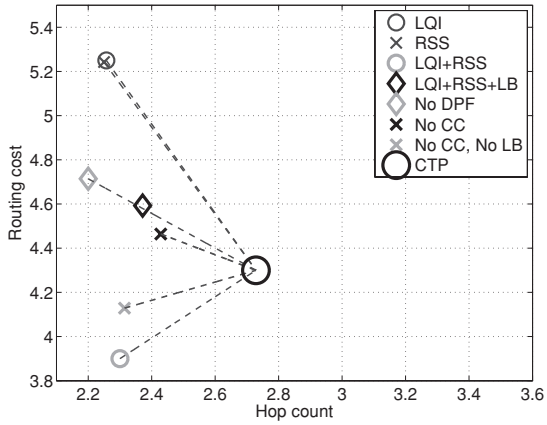


Fig. 11. MoteLab, 130 nodes (December 2008). Relative impact of the components of the Arbutus architecture on the cost plane under light load.

overhead drops to just 5%, only 20% of CTP's overhead. This happens because heavy load conditions are likely to cause disruptions that get CTP to reduce its beaconing interval. Arbutus's uncapped duplicate suppression scheme is very effective, although capped signature caching (similarly to CTP) would need to be employed in larger networks ($|\mathcal{N}| > 200$).

4.5 Relative Impact of the Main Components

In this section, we isolate the impact of load balancing, link estimation, and congestion control on the performance with a set of experiments that we ran on MoteLab in December 2008 (with an average of 130 active nodes). In this set of experiments, we employ CTP's September 2008 version with Four-Bit Link Estimation [Fonseca et al. 2007]. Figures 10 and 11 show the relative impact

of the various components of Arbutus under light load and compare them to CTP on, respectively, the performance plane and the cost plane.

Load balancing. We begin by comparing Arbutus with load balancing ($w_{\mathrm{RSS}} = 1$, $w_{\mathrm{LQI}} = 1$, $w_{\mathrm{B}} = 1$ in Eq. (5), indicated as LQI+RSS+LB in Figures 10 and 11) and Arbutus without load balancing ($w_{\mathrm{RSS}} = 1$, $w_{\mathrm{LQI}} = 1$, $w_{\mathrm{B}} = 0$, indicated as LQI+RSS). We have seen that Arbutus with load balancing works very well under high load thanks to the interplay between load balancing and congestion control, but we have also seen that load balancing reduces Arbutus's goodput under light load, because suboptimal links require more transmissions, and their use therefore limits the goodput and increases the routing cost. Figure 10 shows that, if load balancing is disabled, Arbutus outperforms CTP's goodput even under light load. Figure 11 shows that, under light load, load balancing also has a significant impact on the routing cost, which drastically drops if load balancing is shut down.

Link estimation. We now shut down load balancing and focus on link estimation, which has two main components: CSI estimates from control beacons (beacon-based link estimation) and data plane feedback. We begin to focus on beacon-based link estimation, isolating RSS-only link estimation ($w_{\mathrm{RSS}} = 1$, $w_{\mathrm{LQI}} = 0$, $w_{\mathrm{B}} = 0$, indicated as RSS in Figures 10 and 11), LQI-only link estimation ($w_{\mathrm{RSS}} = 0$, $w_{\mathrm{LQI}} = 1$, $w_{\mathrm{B}} = 0$, indicated as LQI), and their combined effect, which is Arbutus without load balancing ($w_{\mathrm{RSS}} = 1$, $w_{\mathrm{LQI}} = 1$, $w_{\mathrm{B}} = 0$, indicated as LQI+RSS). It is clear from Figure 10 that the joint use of both RSS and LQI makes a big difference, because RSS and LQI complement each other: the former provides soft information about good links, while the latter provides soft information about bad links [Puccinelli and Haenggi 2008b]. Using only LQI yields a higher delivery ratio than using only RSS at the price of a lower goodput; this is because if only RSS is used, the very best links are consistently chosen, which leads to higher goodput but, at the same time, more congestion-induced packet loss. On the other hand, if only LQI is used, there is no soft information about good links: any high LQI link can be selected. It has been shown that the time average of the LQI is a lot more reliable than just a single value. Arbutus does use a single LQI value, but normally compensates it with the concurrent use of RSS. If single-value LQI is the only link metric, however, not-so-good links get chosen based on one high LQI sample, thus causing a significant goodput degradation. Reliability, however, is not affected, since Arbutus uses unconstrained retransmissions, and the fact that any high-LQI link can be arbitrarily chosen means that congestion is not an issue. In other words, the choice of CSI affects the goodput, while the data-driven feedback affects the delivery rate. From the standpoint of cost, Figure 11 shows that using only RSS or only LQI is significantly more costly than combining them, which confirms that using both is extremely beneficial. Note that the same delivery ratio is achieved with RSS and LQI, which are two complementary forms of CSI, and this shows that the delivery performance is not sensitive to the inner cost field metric, which depends on the data-driven feedback.

We now separate beacon-based link estimation from data plane feedback by shutting down the latter ($w_{\mathrm{RSS}} = 1$, $w_{\mathrm{LQI}} = 1$, $w_{\mathrm{B}} = 1$ in Eq. (5), and $y(M_i) = 0$ for all i in Eq. (1), indicated as No DPF), which means that beacon-based estimates are accepted at face value without data-driven feedback. Due to the use of unconstrained retransmissions and congestion control, the lack of data plane feedback hardly affects reliability, but goodput is heavily affected, as shown in Figure 10. Not using data plane feedback means being exposed to link estimation errors and, therefore, suboptimal links; indeed, Figure 11 shows that, in the absence of data plane feedback, Arbutus's routes are shorter than usual, but its routing cost is higher. With blind beacon-based estimates, Arbutus's long hop approach backfires, because its link estimator chooses long links no matter how lossy they are.

Congestion control. We have already stated that Arbutus outperforms CTP owing to long-hop routing and the combined action of congestion control and unconstrained retransmissions. Though CTP has a congestion detection mechanism and can employ existing congestion control schemes, it does not perform congestion control. It is therefore fair to ask to what extent congestion control boosts Arbutus's performance. We therefore disable the congestion control state machine (by setting $W_{\mathrm{c}} > 1$) as well as congestion onset detection both in the presence of load balancing ($w_{\mathrm{RSS}} = 1$, $w_{\mathrm{LQI}} = 1$, $w_{\mathrm{B}} = 1$ in Eq. (5), indicated as No CC) and in its absence ($w_{\mathrm{RSS}} = 1$, $w_{\mathrm{LQI}} = 1$, $w_{\mathrm{B}} = 0$ in Eq. (5), indicated as No CC, No LB).

Figure 10 shows that, under light load, with load balancing and no congestion control, Arbutus's delivery ratio matches CTP's, while its goodput drops below CTP's. Nonetheless, if we also shut down load balancing, Arbutus's delivery rate and goodput almost match their values in the case of Arbutus with congestion control and without load balancing. The same is true for the cost plane shown in Figure 11: with load balancing and no congestion control, both the routing cost and the hop count increase, but with no load balancing the cost drops well below CTP's, though it is still a bit larger than in the case of Arbutus with congestion control and no load balancing. This result confirms the key role of the interplay between load balancing and congestion control and also underlines that Arbutus's main strength does not lie in its congestion control scheme, but in its long-hop routing strategy, made possible by its dual cost field link estimation scheme. This is also confirmed by the results under high load, shown in Figure 12, where we consider Arbutus without load balancing (LQI+RSS), with load balancing (LQI+RSS+LB), and without congestion control (No CC). Under high load, congestion control becomes a lot more crucial: without it, Arbutus's delivery ratio drops below 70%; nevertheless, Arbutus with no congestion control still outperforms CTP, providing further evidence that the main strength of Arbutus lies in its dual cost field and the low-depth routing tree that it leads to. Note that, in the absence of congestion control, Arbutus achieves a better goodput. This is a clear indication that the backpressure scheme is rather conservative; indeed, a congested node asks its children to simply stop sending. We have experimented with more lenient backpressure policies, such as asking children to slow down rather than just stop, and we

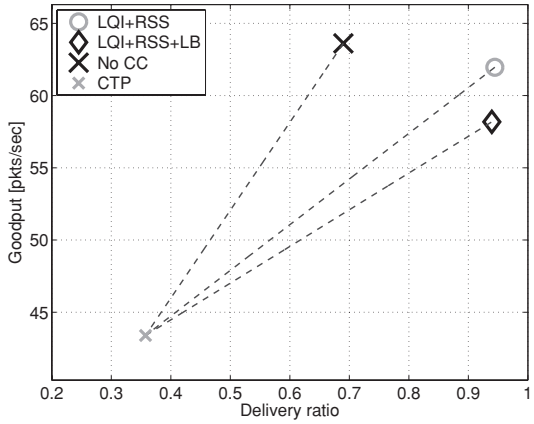


Fig. 12. MoteLab, 130 nodes (December 2008). Relative impact of the components of the Arbutus architecture on the performance plane under heavy load.

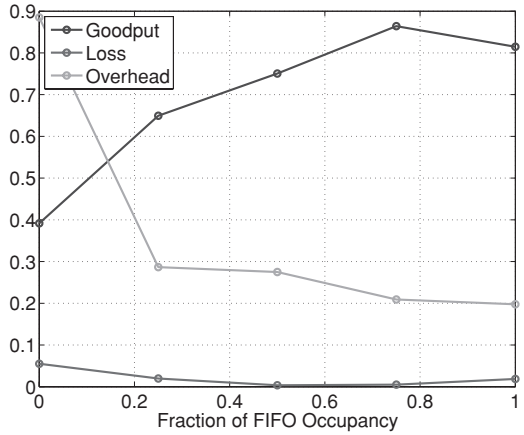


Fig. 13. Our experimental data (under light load) shows that an overly conservative calibration of W_c causes the system to overreact and flood the network with control traffic.

have found that they provide a small goodput gain at the price of a significant degradation of the delivery ratio. Figure 12 also confirms the interplay between congestion control and load balancing, which minimizes the impact of load balancing on the goodput under heavy load (even with load balancing, Arbutus outperforms CTP under heavy load).

We will now provide a justification for our calibration of W_c, the key threshold for the congestion control scheme. The threshold W_c can be calibrated empirically based on the experimental data shown in Figure 13, obtained by averaging the goodput, packet loss, and control overhead results obtained with Arbutus under light load for different topologies. These results were obtained on MoteLab under light load in June 2009 with about 100 nodes; the goodput is normalized with respect to the target offered load (0.1 pkt/sec), and the

Table IV. MoteLab, 130 Nodes

Performance Indicator	Arbutus (LB)	Arbutus (no LB)	CTP (4BLE)
Delivery ratio	0.98 ± 0.02	0.99 ± 0.01	0.93 ± 0.02
Goodput [pkts/sec]	11.44 ± 1.17	11.96 ± 0.65	11.70 ± 0.26
Hop count	2.43 ± 0.42	2.35 ± 0.37	2.85 ± 0.67
Routing cost	4.73 ± 1.44	4.03 ± 1.11	4.45 ± 1.03
Duplicate suppression	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01

Overall results for the winter 2008 experiments, averaged over all the topologies that we considered, under light load. For each performance indicator, we provide the mean μ and the standard deviation σ in the form $\mu \pm \sigma$. We compare Arbutus with load balancing (LB), Arbutus without load balancing (no LB), and CTP with Four-Bit Link Estimation (4BLE).

Table V. MoteLab, 130 Nodes

Performance Indicator	Arbutus (LB)	Arbutus (no LB)	CTP (4BLE)
Delivery ratio	0.94 ± 0.04	0.96 ± 0.05	0.48 ± 0.02
Goodput [pkts/sec]	59.80 ± 9.10	63.70 ± 9.10	44.20 ± 2.60
Hop count	2.19 ± 0.38	2.04 ± 0.29	3.35 ± 0.13
Routing cost	4.27 ± 0.47	4.20 ± 0.67	6.60 ± 1.17
Duplicate suppression	0.99 ± 0.01	0.99 ± 0.01	0.91 ± 0.05

Overall results for the winter 2008 experiments, averaged over all the topologies that we considered, under heavy load. For each performance indicator, we provide the mean μ and the standard deviation σ in the form $\mu \pm \sigma$. We compare Arbutus with load balancing (LB), Arbutus without load balancing (no LB), and CTP with Four-Bit Link Estimation (4BLE).

absolute goodput can be obtained by multiplying the normalized goodput by 10 (100 nodes * 0.1 pkt/sec). If an overly conservative calibration is chosen, the system overreacts to congestion, the control overhead blows up, and a severe performance degradation occurs. A relaxed calibration, on the contrary, defies the purpose of congestion control and leaves the system vulnerable to congestion-induced losses. Based on these results, our implementation employs $W_c = 0.75$ (if the queue is 75% full, the system signals congestion).

Overall benchmarking. We now focus on the results with and without load balancing, and examine how they compare to CTP with Four-Bit Link Estimation. Tables IV and V present an overview of our December 2008 results on the 130-node MoteLab testbed, respectively for light and heavy load.

Table IV confirms that, while Arbutus is more reliable (both with and without load balancing), CTP outperforms Arbutus with load balancing in terms of goodput under light load. As already seen in Figures 10 and 11, load balancing is responsible for the performance degradation remarked in Table III. In fact, without load balancing Arbutus outperforms CTP also in terms of goodput. Load balancing also has an impact on the routing cost: without load balancing, Arbutus achieves a lower cost and tree depth than with load balancing.

Table V shows the results under heavy load. As already shown in Figure 12, load balancing has a negligible cost under high load, and this is true in all dimensions (delivery ratio, goodput, hop count, routing cost). On the CTP side, a comparison between Tables III and IV–V shows that Four-Bit Link Estimation significantly outperforms LEEP in terms of routing cost. We also note that the

Table VI. MoteLab, about 100 Nodes

Performance Indicator	Arbutus - Unconstrained	Arbutus - Constrained
Delivery ratio	0.99 ± 0.01	0.97 ± 0.12
Goodput [pkts/sec]	7.11 ± 2.88	7.20 ± 1.98
Hop count	2.80 ± 1.19	2.76 ± 1.17
Routing cost	4.35 ± 1.88	3.63 ± 1.53

Overall results for the June 2009 experiments, averaged over all the topologies that we considered, under light load. For each performance indicator, we provide the mean μ and the standard deviation σ in the form $\mu \pm \sigma$. We compare Arbutus without load balancing with unconstrained retransmissions (Unconstrained) and constrained retransmissions (Constrained) with $N_{\max} = 30$.

September 2008 version of CTP performs much better than the February 2008 version in terms of duplicate suppression.

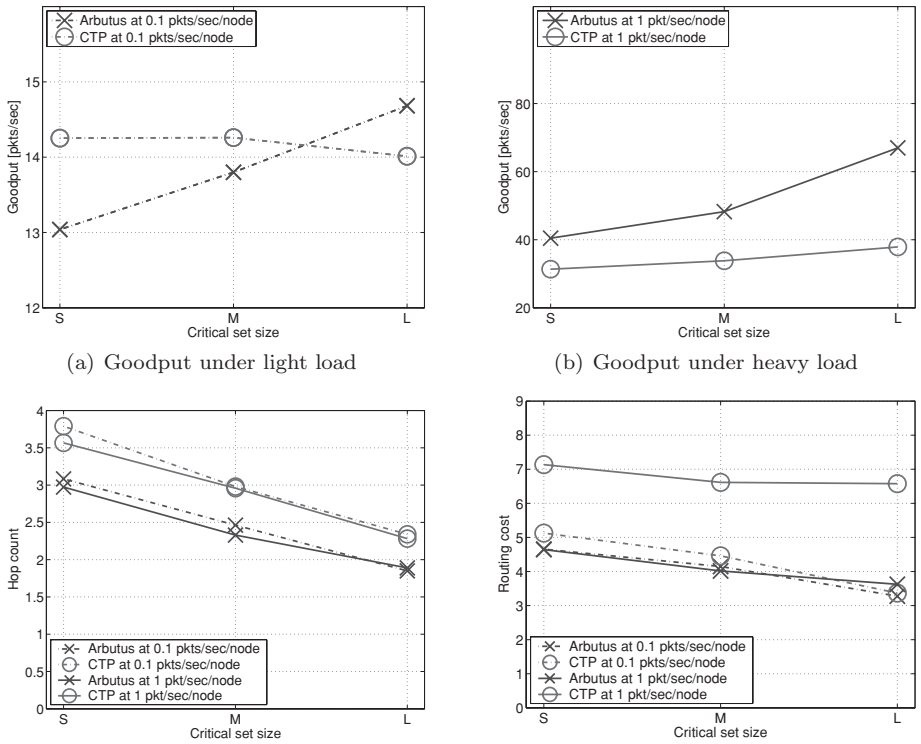
So far, we have only considered Arbutus with unconstrained retransmissions. We have run a number of experiments, with several different sink assignments, to quantify the impact of constrained retransmissions on the performance. We set $N_{\max} = 30$, just like CTP. Interestingly, we found that constrained retransmissions do not, on average, increase the goodput. They do cause some packet loss, but typically not more than an average 3%. One strong advantage of constrained retransmissions, however, is energy conservation, as shown by the significant reduction of the average routing cost. The results of these experiments, run in June 2009 on MoteLab (with about 100 active nodes) are in Table VI.

5. THE IMPACT OF NETWORK TOPOLOGY ON ROUTING

We have seen in Section 4 that network topology has a significant impact on routing performance. In this section, we further characterize this impact. We study the impact of the critical set size in Section 5.1 and analyze the role of transitional links in Section 5.2.

5.1 Impact of the Critical Set Size

Since we employ a many-to-one collection tree, the critical set size is the most natural way to quantify a given topology. As already remarked, in our experiments on Twist we observed a much less significant topology-induced variability compared to MoteLab. The reason for this is that Twist has a very regular layout and a relatively high node degree, while MoteLab has a more irregular, complex topology, and a lower average node degree. All the Twist sink assignments that we considered have a critical set size between 40 and 60, which is large compared to Twist's network size (between 95 and 100). Both protocols perform much better in Twist (Figure 6) than they do in MoteLab (Figure 5) showing that a large critical set is extremely beneficial. In Twist, under heavy load, Arbutus achieves an average normalized goodput (i.e., normalized with respect to the target offered load) of 0.75, 2.2 times higher than in MoteLab. CTP also performs much better: 0.61 in Twist and 0.22 in MoteLab. A large critical set size has an even more significant impact on the routing cost. Under heavy load, Arbutus goes from 4.05 in MoteLab to 2.16 in Twist, while under



(a) Goodput under light load

(b) Goodput under heavy load

(c) Hop count

(d) Routing cost

Fig. 14. Overview of the routing performance of Arbutus with load balancing: the results shown herein are averaged over all topologies with a critical set size within one of three classes: small (*S*, less than 20 critical nodes), medium (*M*, 20 to 40 critical nodes), and large (*L*, over 40 critical nodes).

light load, it goes from 4.25 in MoteLab to 1.68 in Twist. The larger the critical set, the easier it is to get more packets to more nodes (higher goodput) with fewer transmissions per delivery (lower cost). A performance degradation can be expected to occur in topologies with a small critical set, which are more likely to suffer from severe congestion. In fact, if only a few critical nodes are relaying traffic from many upstream peers, interference and congestion are unavoidable. The problem with a similar scenario is that congestion causes more interference, because congestion management requires additional control traffic that interferes with the intense data traffic.

Given its complex layout, MoteLab is the testbed that best lends itself to the study of the impact of the topology. Figure 14 provides an overview of the joint impact of the offered load and the topology on the goodput, coverage, hop count, and routing cost in all the March–April 2008 experiments on MoteLab (155 nodes), where we employed Arbutus with load balancing and CTP with LEEP. The results for light and heavy load are averaged over all topologies within a given interval of critical set sizes. Given that there are 155 nodes, we classify those critical sets with less than 20 critical nodes as small (indicated

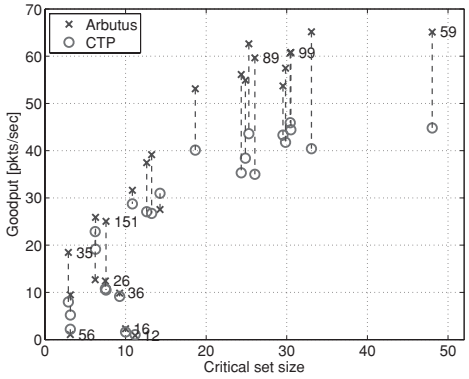


Fig. 15. MoteLab, 90 nodes: goodput as a function of the critical set size.

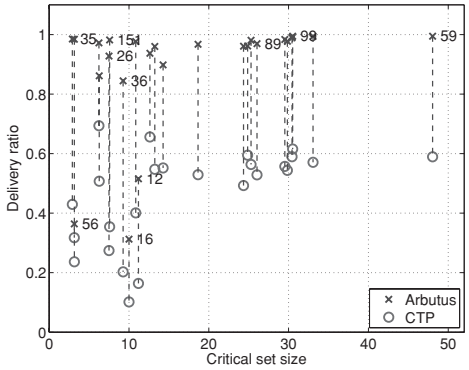


Fig. 16. MoteLab, 90 nodes: reliability as a function of the critical area size.

as S), those between 20 and 40 as medium (M), and those with over 40 as large (L). We do not consider the delivery ratio because we have seen in Section 4.4 that it is not affected by the topology.

Figures 14(a) (goodput under light load) and 14(b) (goodput under heavy load) confirm that, as we remarked in Section 4.4, Arbutus's goodput leverages on the topology: due to its long-hop routing approach, Arbutus performs much better if the critical set size is large. This is particularly true under light load, where load balancing takes a toll on the goodput for small and medium critical set sizes. The critical set size also significantly affects hop count (Figure 14(c)) and routing cost (Figure 14(d)), both of which are relatively insensitive to the offered load in the case of Arbutus, as seen in Section 4.4.

Figures 15 and 16 show, respectively, the goodput and delivery ratio results under heavy load for various values of critical set size in March 2008's 90-node MoteLab testbed. In these experiments, we used Arbutus with load balancing and CTP with LEEP. Particular effects can be observed in the 90-node testbed due to its much sparser connectivity. In particular, sinks with very low degrees

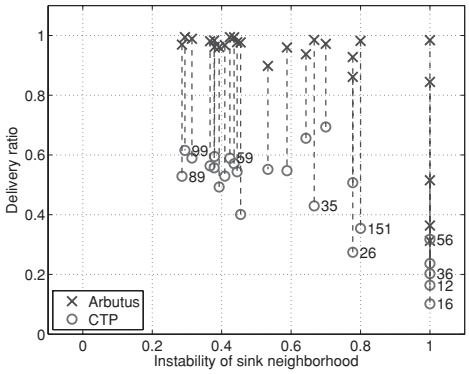


Fig. 17. MoteLab, 90 nodes. Neighborhood instability is a good predictor of performance degradation. Topologies 12, 16, and 56 exhibit a very poor performance with both protocols.

have a rather unpredictable performance not only in terms of goodput, but also in terms of delivery ratio. There are some significant outliers in Arbutus's performance: 12, 16, and 56 are truly awful for both protocols. They all have small degrees, but so do many other sink assignments that do not experience the total collapse suffered by these three outliers. This suggests that, in conditions of sparse connectivity, critical set size does not fully capture the impact of the topology. There exists another significant topological effect, the presence of transitional links, which we discuss in Section 5.2.

5.2 Impact of Transitional Links

Links in the transitional region are affected by a lack of long-term stability. We define *neighborhood instability* based on the fraction of links in the transitional region, that is, the links whose quality lies below a given RSS threshold (which we set at -87dBm, following Srinivasan and Levis [2006]) and a given LQI threshold (which we set to 100, based on empirical observations). Figure 17 shows that, in the 90-node MoteLab testbed, sink assignments with a poor delivery rate performance have instability values close or equal to 1: most of their links are asymmetric or just poor in both directions. If the sink has almost exclusively links that are poor in both the inbound and the outbound direction, its beacons normally do not make it to its neighbors, and when they do, the data packets from the sink's neighbors normally do not make it to the sink. In this case, unconstrained retransmissions still save the delivery ratio, but the goodput drops dramatically. If links are asymmetric and predominantly lossy in the inbound direction, beacons normally make it to the sink's neighbors, but their packets normally do not make it to the sink; again, unconstrained retransmissions still save the delivery ratio, but the goodput plummets. If the sink has asymmetric links that are predominantly lossy in the outbound direction, the sink's beacons normally get dropped. If the sink's beacons happen to get through, the sink's neighbors attempt to send data packets to the sink. The sink's inbound links are not particularly lossy, and the sink is likely to

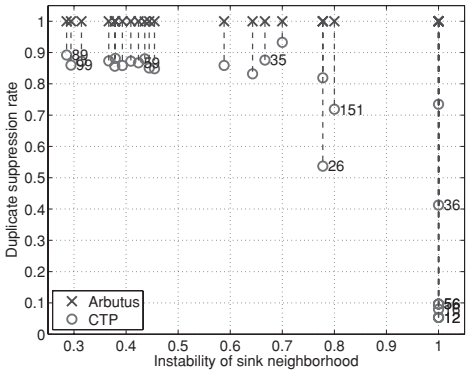


Fig. 18. MoteLab, 90 nodes. Correlation between CTP duplicate suppression rate and instability; 12, 16, and 56 again stand out as outliers.

receive the packets sent by its neighbors, but the sink's ACKs back to its neighbors get dropped over its lossy outbound links. The neighbors send the same packets over and over, and the sink ends up with plenty of duplicates as well as lots of congestion. Therefore, if the sink has asymmetric links that are predominantly lossy in the outbound direction, there is not only a goodput degradation, but also a drop in delivery rate.

Since CTP puts a cap to the number of packet signatures that it caches for duplicate suppression, CTP's duplicate suppression rate is a good indication of this phenomenon, and is indeed much lower for those sink assignments that also see a significant delivery rate degradation, as can be seen in Figures 17 and 18, which show, respectively, the delivery ratio and the duplicate suppression rate against the instability of the sink neighborhood. Those same nodes 12, 16, and 56 that we labeled as outliers in Section 5.1 also stand out in Figures 17 and 18. In particular, Figure 18 shows that there is a definite correlation between CTP's duplicate suppression rate and sink neighborhood instability. CTP has a much harder time suppressing duplicates in unstable topologies, simply because there are many more duplicates to suppress. If Arbutus also capped the number of node signatures that it can store for duplicate suppressions, it would have the exact same problem as CTP. Instability is more likely if the sink has a low critical set size, because in this case there exists a limited number of paths to the sink; this is the condition of the performance outliers in the 90-node MoteLab testbed. Figure 19 shows the duplicate suppression rate against the critical set size (node degree), confirming that sink assignments affected by instability (such as outliers 12, 16, and 56) all have a small critical set size.

Instability is a byproduct of multipath fading [Puccinelli and Haenggi 2006a], whose effect is frequency dependent; indeed, unstable topologies display a wide range of variability across different frequencies. Figure 20 shows the goodput-delivery performance of sink assignment 56 as we modify the 802.15.4 channel employed by the CC2420 transceiver. Channel 26, used in all other experiments in this article, yields the worst performance with sink

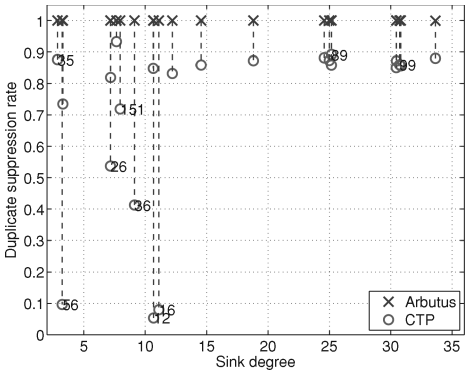


Fig. 19. MoteLab, 90 nodes. Outliers 12, 16, and 56 have both a small critical set and a low CTP duplicate suppression rate, which is a good predictor of instability.

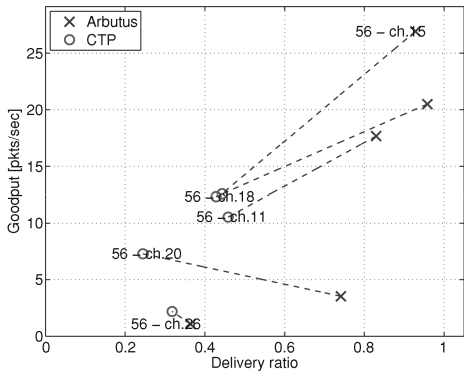


Fig. 20. MoteLab, 90 nodes. Different 802.15.4 with sink assignment 56 produce very different results: for instance, fading is not an issue on channel 11, but is destructive on channel 26.

assignment 56, while channel 15 yields the best performance. This experiment therefore confirms that fading may have a devastating impact on topologies with a small critical set. Interference is also an issue here: for instance, 802.15.4 channels 11 and 18 fully overlap with 802.11 channels 1 and 6.

6. CLOSING REMARKS

Long hops. Our extensive experimental results, obtained on large-scale public testbeds of low-end sensing nodes, provide further evidence in favor of long-hop routing, which we have showed to boost routing performance provided that the routing protocol is supported by a solid link estimator. Our double cost field link estimator employs an outer field based on depth estimates to avoid unnecessarily long routes, and an inner field based on link quality estimates to avoid lossy links. To ensure link estimation robustness, both fields refine beacon-based estimates with data-driven feedback. The double cost field structure keeps the routes as short as possible while avoiding unreliable links. The outer cost minimizes the hop count while also minimizing the routing cost.

Reliability. We have showed that Arbutus achieves a significant degree of reliability; this is mainly due to its active pursuit of long hops, along with the combined action of congestion control and unconstrained retransmissions. We have shown that Arbutus's reliability is not sensitive to the offered load and the network topology.

Scalability. Since Arbutus contains mechanisms that explicitly address a relatively high traffic load, the performance gap between Arbutus and CTP widens as the offered load increases.

Load balancing. In Puccinelli and Haenggi [2008a], we estimate that load balancing extends network lifetime by 30% under heavy load; in Puccinelli and Haenggi [2009]), we confirm this estimate by emulating energy depletion and measuring the lifetime benefit. In this work, however, we find that, under light load, load balancing is responsible for a 5% goodput loss. While Arbutus with load balancing yields a goodput that is 3% below CTP's, Arbutus without load balancing improves CTP's goodput by 2%; this happens because load balancing requires the occasional use of suboptimal links. Under heavy load, however, there is an interplay between load balancing and congestion control, because part of the goodput loss caused by load balancing is partially compensated for by the decreased congestion.

Topology. Our results show that the network topology has a huge impact on routing performance. Even with the same network, performance can change dramatically if we modify the sink placement. While in actual deployments there is often a limited amount of freedom in node placement and sink selection, care should nonetheless be exercised to ensure enough route diversity near the sink by way of a relatively large critical set (compared to network size).

Exploring the design space. The Arbutus architecture lends itself to further explorations of the routing design space. In particular, an adaptation of Arbutus for point-to-point routing over a mesh is also possible: a node would have to keep state for all its intended destinations, which would be feasible due to the fact that Arbutus saves memory by not relying on a routing table.

Impact of the MAC layer. In this study, we have bulit on top of the standard CSMA-based MAC layer of TinyOS 2.x. It would be of great interest to study the performance of Arbutus on top of different MAC schemes. In particular, the use of a prioritized MAC [Woo and Culler 2001] that would shorten the backoff for congested nodes would probably increase Arbutus's goodput.

Low-power operation. A future study could be devoted to investigating the operation of Arbutus on top of the Low Power Listeningprotocol (LPL) [Polastre et al. 2004] or Low Power Probing [Musaloiu-E. et al. 2008]. A version of CTP with LPL is already available, and it would again serve as a benchmark. It would be of particular interest to use Arbutus for backbone routing in a mostly-off sensor network where the use of sleep modes is widespread.

ACKNOWLEDGMENTS

A special thanks goes to Omprakash Gnawali for providing lots of invaluable feedback on our work. We also thank him, along with Ramesh Govindan, for providing us access to their Tutornet testbed at USC. Many thanks go to Geoff Werner Challen and Matt Welsh for making their MoteLab testbed at Harvard available to the community. We would like to thank Vlado Handziski and Adam Wolisz for providing us access to their Twist testbed at TU Berlin, and we would like to thank David Gay for providing us access and getting us set up to work on the Mirage testbed at Intel Berkeley.

REFERENCES

CERPA, A., WONG, J., POTKONJAK, M., AND ESTRIN, D. 2005. Temporal properties of low power wireless links: Modeling and implications on multi-hop routing. In *Proceedings of the 4th ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN'05)*.

CHOI, J., LEE, J., WACHS, M., CHEN, Z., AND LEVIS, P. 2007a. Fair waiting protocol: Achieving isolation in wireless sensornets. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*.

CHOI, J., LEE, J., WACHS, M., CHEN, Z., AND LEVIS, P. 2007b. Opening the Sensornet black box. In *Proceedings of the International Workshop on Wireless Sensornet Architecture (WWSNA'07)*.

COUTO, D. D., AGUAYO, D., BICKET, J., AND MORRIS, R. 2003. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom'03)*.

FILIPPONI, L., SANTINI, S., AND VITALETTI, A. 2008. Data collection in wireless sensor networks for noise pollution monitoring. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS'08)*.

FONSECA, R., GNAWALI, O., JAMIESON, K., AND LEVIS, P. 2007. Four-Bit wireless link estimation. In *Proceedings of the 6th Workshop on Hot Topics in Networks (HotNets-VI)*.

GANESAN, D., GOVINDAN, R., SHENKER, S., AND ESTRIN, D. 2002. Highly resilient, energy efficient multipath routing in wireless sensor networks. *ACM Mobile Comput. Comm. Rev. 1*, 2, 10–24.

GNAWALI, O. 2007. The link estimation exchange protocol (LEEP). `http://www.tinyos.net/tinyos-2.x/doc/html/tep124.html`.

GNAWALI, O., FONSECA, R., JAMIESON, K., MOSS, D., AND LEVIS, P. 2009. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*.

GNAWALI, O., YARVIS, M., HEIDEMANN, J., AND GOVINDAN, R. 2004. Interaction of retransmission, blacklisting, and routing metrics for reliability in sensor network routing. In *Proceedings of the 1st IEEE Conference on Sensor and Ad Hoc Communication and Networks (SECON'04)*. 34–43.

HAENGGI, M. AND PUCCINELLI, D. 2005. Routing in ad hoc networks: A case for long hops. *IEEE Comm. Mag. 43*, 93–101.

HANDZISKI, V., KOEPKE, A., WILLIG, A., AND WOLISZ, A. 2006. TWIST: A scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN'06)*.

HAUER, J., HANDZISKI, V., KOEPKE, A., WILLIG, A., AND WOLISZ, A. 2008. A component framework for content-based publish/subscribe in sensor networks. In *Proceedings of the IEEE European Workshop on Wireless Sensor Networks (EWSN'08)*.

HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. 2000. System architecture directions for network sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*.

HULL, B., JAMIESON, K., AND BALAKRISHNAN, H. 2004. Mitigating congestion in wireless sensor networks. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*.

JAIN, R. 1991. *The Art of Computer Systems Performance Analysis*. Wiley.

KARENOS, K. AND KALOGERAKI, V. 2007. Facilitating congestion avoidance in sensor networks with a mobile sink. In *Proceedings of the 28th International IEEE Real-Time Systems Symposium (RTSS'07)*.

KOTHARI, N., MILLSTEIN, T., AND GOVINDAN, R. 2008. Deriving state machines from TinyOS programs using symbolic execution. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN'08)*.

LANGENDOEN, K., BAGGIO, A., AND VISSER, O. 2006. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'06)*.

MUSALOIU-E., R., LIANG, C., AND TERZIS, A. 2008. Deriving state machines from TinyOS programs using symbolic execution. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN'08)*.

PAEK, J. AND GOVINDAN, R. 2007. RCRT: Rate-Controlled reliable transport for wireless sensor networks. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*.

POLASTRE, J., HILL, J., AND CULLER, D. 2004. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*.

POOR, R. 2000. Gradient routing in ad hoc networks. `http://www.media.mit.edu/pia/Research/ESP/texts/poorieeepaper.pdf`.

PUCCINELLI, D. AND HAENGGI, M. 2006a. Multipath fading in wireless sensor networks: Measurements and interpretation. In *Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC'06)*.

PUCCINELLI, D. AND HAENGGI, M. 2006b. Spatial diversity benefits by means of induced fading. In *Proceedings of the 3rd IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)*.

PUCCINELLI, D. AND HAENGGI, M. 2008a. Arbutus: Network-Layer load balancing for wireless sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'08)*.

PUCCINELLI, D. AND HAENGGI, M. 2008b. DUCHY: Double cost field hybrid link estimation for low-power wireless sensor networks. In *Proceedings of the 5th Workshop on Embedded Networked Sensors (HotEmNets'08)*.

PUCCINELLI, D. AND HAENGGI, M. 2009. Lifetime benefits through load balancing in homogeneous sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'09)*.

RAMAKRISHNAN, K. AND JAIN, R. 1990. A binary feedback scheme for congestion avoidance in computer networks. *ACM Trans. Comput. Syst. 8*, 2 (May), 158–181.

RANGWALA, S., GUMMADI, R., GOVINDAN, R., AND PSOUNIS, K. 2006. Interference-Aware fair rate control in wireless sensor networks. In *Proceedings of the ACM SIGCOMM Symposium on Network Architectures and Protocols*.

SRINIVASAN, K., DUTTA, P., TAVAKOLI, A., AND LEVIS, P. 2008a. An empirical study of low-power wireless. Tech. rep. SING-08-03, Stanford University.

SRINIVASAN, K., KAZANDJIEVA, M., AGARWAL, S., AND LEVIS, P. 2008b. The beta-factor: Improving bimodal wireless networks. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*.

SRINIVASAN, K. AND LEVIS, P. 2006. RSSI is under appreciated. In *Proceedings of the 3rd Workshop on Embedded Networked Sensors (EmNets'06)*.

WACHS, M., CHOI, J., LEE, J., SRINIVASAN, K., CHEN, Z., JAIN, M., AND LEVIS, P. 2007. Visibility: A new metric for protocol design. In *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*.

WAN, C., EISENMAN, S., AND CAMPBELL, A. 2003. CODA: Congestion detection and avoidance in sensor networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor systems (SenSys'03)*.

WAN, C., EISENMAN, S., CAMPBELL, A., AND CROWCROFT, J. 2007. Overload traffic management in sensor networks. *ACM Trans. Sensor Netw. 3*, 4.

WANG, Q., HEMPSTEAD, M., AND YANG, W. 2006. A realistic power consumption model for wireless sensor network devices. In *Proceedings of the 3rd IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)*.

WERNER-ALLEN, G., SWIESKOWSKI, P., AND WELSH, M. 2005. MoteLab: A wireless sensor network testbed. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*.

WOO, A. 2004. A holistic approach to multihop routing in sensor networks. Ph.D. thesis, University of California at Berkeley.

WOO, A. AND CULLER, D. 2001. A transmission control scheme for media access in sensor networks. In *Proceedings of the 7th International Conference on Mobile Computing and Networking (MobiCom'01)*.

WOO, A., TONG, T., AND CULLER, D. 2003. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*.

ZHANG, H., SANG, L., AND ARORA, A. 2008. Data-Driven link estimation in sensor networks: An accuracy perspective. Tech. rep. DNC-TR-08-02, Wayne State University.

ZHAO, J. AND GOVINDAN, R. 2003. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*.

ZUNIGA, M. AND KRISHNAMACHARI, B. 2007. An analysis of unreliability and asymmetry in low-power wireless links. *ACM Trans. Sensor Netw. 3*, 2, 1–30.

Received June 208; revised July 2009; accepted September 2009